RIVM rapport 773401005/2003

Reference Guide Microsoft.NET

M van der Zee, G Verspaij, S Rosbergen

Intern rapport

Dit onderzoek werd verricht in opdracht en ten laste van LAE-RIS, in het kader van project M773401, Bronnen.

RIVM, Postbus 1, 3720 BA Bilthoven, telefoon: 030 - 274 91 11; fax: 030 - 274 29 71

Abstract

Described here is the Microsoft .NET technology and its application in the Office for Environmental Assessment (MNP). This report was written with the Microsoft.NET technology development tools, making use of the knowledge taken from the Internet 'white papers'.

A .NET pilot project, aimed at accumulating more knowledge on Microsoft .NET technology, was carried out simultaneously with the production of the report. The pilot's main focus was on Web applications.

All new applications will be Web enabled within the MNP department. Therefore, this report will concentrate mostly on Web applications and it's architecture.

J2EE (Java) and .NET (Microsoft) are the new technologies to develop Web applications at this moment. Because the formerly department LAE has developed many applications with VB(A), this report will describe the new .NET technology only.

Overall
According to Microsoft's vision, every user must be able to control his or her information independent of which device he or she uses. Microsoft has developed the .NET technology to realise this vision. The new architecture of the .NET technology consists of 'Smart Clients', 'Web Services', servers and developer tools. Use of this technology enables to obtain central data independent of the device.

Smart Clients are devices, which are capable of collecting, understanding and displaying data using the .NET technology. Devices can communicate and exchange data with Web Services. Servers are used to store data. Visual Studio.NET is a developer tool for building .NET applications. It includes all Microsoft .NET languages and the .NET Framework. The .NET Framework consists of the Common Language Runtime, necessary to use .NET applications, and several Class Libraries. A Class Library is an extensive collection of useful compiled code.

A .NET executable or DLL is called an assembly. They solve the DLL-hell. Besides that they make distribution of .NET applications less complicate. Desktop applications need the .NET Framework to run properly. Distribution of Web applications is even more simply. It is only necessary to install the application and the .NET Framework on a web server.

Architecture
Web Services can be used for communications between a Desktop application and remote DLL's. The possibility to let devices communicate via the Internet is a great advantage of Web Services. This is a perfect solution for distributed applications which have to be used outside the RIVM network environment.

Applications can be split into different assemblies, so-called n-tier infrastructure. Each assembly will have its own task (e.g. User Interface, Business Rules and data logic). Since this will mean a different way of programming for most developers, the .NET developer should be trained in multiple disciplines. Advantage of the n-tier infrastructure is that developers can accomplish different roles in a n-tier project. Scalability and maintenance of the application are other advantages of n-tier infrastructure.

Choice of development language
VB.NET is the proper language for making a Desktop application, at least if chosen by an experienced VB6 developer. The language choice for a C or Java developer will probably be C#. This is a personal preference only. The syntax is the only difference between both languages. The developer uses Windows Forms with VB.NET, while ASP.NET is more obvious when developing an User Interface for Web applications. In this case, the User Interface will be developed with Web Forms. The language developed for the ASP.NET pages may also be VB.NET. Both Window Forms and Web Forms contains many controls to build a rich User Interface.

Visual Basic.NET
Visual Basic .NET (VB.NET) will be recognised by many experienced VB developers. However, major changes have been made to VB.NET, the most important being its complete object orientation. This does not only affect the way code is written, but also the choices for the application structure. Furthermore, there are some changes in data types and how types are declared. The way errors are dealt with in VB.NET is completely new. There is also a wizard for conversion of VB6 applications to VB.NET. However, the usefulness of this wizard will have to be considered per project, since there will still be code that cannot be converted by the wizard.

Finally, the .NET Framework consists of much functional code, making it important to search the Framework for codes before programming a function.

Active Server Pages.NET
ASP.NET is succeeding ASP as a develop environment for building Web applications. With ASP.NET it is also possible to build Web Services. The code of a ASP.NET application will be executed on a web server, which will generate a HTML page for the user. Contrary to ASP there are a lot of web controls in ASP.NET with which a developer can make a rich User Interface.

Web applications can be developed with the same languages as those for developing Desktop applications. It is not necessary to use Microsoft .NET languages, even languages like Delphi or Fortran can be used. This can be considered as a great advantages, because it is not necessary or developers to learn new languages. The ASP.NET code will be stored separated in so-called Code Behind files which results in a clear separation between User Interface and code is provided by this addition.

Hardware and software
The system requirements for developer PCs have increased for Visual Studio .NET. In practice, Visual Studio.NET has been shown to work well with Windows 2000 or higher. However the MNP department still uses Windows NT4 as operating system for most of it's workstations at this moment.

The .NET Framework is necessary to be able to run .NET applications. The .NET Framework should be installed on every workstation for Desktop applications. For Web applications it is sufficient to install the .NET Framework on the web server. Therefore, it is not necessary for Web applications to install the .NET Framework on each workstation. This is a great advantage. A free copy of the .NET Framework can be downloaded from the Microsoft website. Although web applications work with all browser types, they are fully supported by Internet Explorer 6.

Conclusions

Microsoft.NET is fully accepted in the ICT world. Not only Microsoft's products like Windows, Office and SQL Server, but also Unix and Oracle will (soon) support the .NET technology. The .NET technology will therefore fit very well in the MNP department's infrastructure and legacy systems.

Microsoft.NET offers many possibilities. Because of its very extensive Framework and its support of many development languages, Microsoft.NET has proven to be very useful tool for MNP. It is also very easy to create a rich User Interface for both Desktop applications and Web applications.

With assemblies not only DLL-hell has been solved, also distribution of .NET applications is made simpler. Both have great advantages for the maintenance and deployment of applications.

Microsoft.NET is one of the better solutions, especially for new applications. It's still important to consider between the advantages of .NET and the investment for converting existing applications. A conversion of a large application will cost generally as much time as the time to rebuild the application in .NET from scratch. To convert an existing application to .NET is a good idea when a Web User Interface is desired.

Investment in knowledge is necessary to profit from the advantages of .NET. These advantages will lead to better, more scalable and more maintainable applications.

Samenvatting

Dit rapport beschrijft de Microsoft .NET (spreek uit: dot NET) technologie en de mogelijke toepassingen hiervan voor het MNP. Dit rapport is tot stand gekomen door onderzoek met behulp van de ontwikkeltools van Microsoft .NET en met whitepapers van het Internet.

Tegelijkertijd met het opstellen van dit document is een .NET pilot uitgevoerd om meer inzicht te krijgen in de .NET technologie. Deze pilot was vooral toegespitst op Web applicaties. De resultaten van de pilot zijn verwerkt in de conclusies.

Binnen het MNP wordt het standaard om alle nieuwe applicaties Web enabled te maken. Mede hierom wordt er in deze Reference Guide vooral aandacht gegeven aan Web applicaties en de benodigde architectuur.

Op dit moment zijn er twee stromingen in nieuwe technologieën waarmee Web applicaties gerealiseerd kunnen worden, te weten J2EE (Java) en .NET (Microsoft). Doordat een groot aantal applicaties van het voormalige LAE ontwikkeld zijn met VB(A) wordt in dit document de nieuwe technologie van Microsoft onder de loep genomen.

Algemeen
Microsofts visie is dat iedereen op een willekeurig apparaat over zijn of haar informatie kan beschikken. Om deze visie te realiseren heeft Microsoft de .NET technologie ontwikkeld. De nieuwe architectuur van de .NET technologie bestaat uit ‘Smart Clients’, ‘Web Services’, Servers en Developer tools. Hiermee is het mogelijk om onafhankelijk van het gebruikte apparaat (PC, handheld, GSM etc.) gegevens te delen. Dit is weer onafhankelijk van waar de data zich bevindt.

Onder Smart Clients worden apparaten verstaan die zelfstandig met behulp van de .NET technologie data kunnen ophalen, interpreteren en tonen. Web Services is een applicatielogica waarmee verschillende apparaten met elkaar kunnen communiceren en data kunnen uitwisselen. De Servers zijn de apparaten die zorgen voor opslag van data. Visual Studio.NET is de developer tool voor het bouwen van .NET applicaties. Het bevat alle Microsoft ontwikkeltalen voor .NET (VB.NET, C# en C++) en het .NET Framework. Het .NET Framework is benodigd om .NET applicaties te kunnen gebruiken en bevat daarnaast een zeer uitgebreide ‘bibliotheek’ met gecompileerde code.

Een .NET executable of DLL wordt ook wel een assembly genoemd. Door de komst van assemblies is de DLL-Hell opgelost. Daarnaast is ook de distributie eenvoudiger geworden. Bij Desktop applicaties is echter wel het Framework nodig om de applicatie te kunnen gebruiken. Bij Web applicaties is de distributie nog eenvoudiger. Hier hoeft de applicatie en het Framework slechts op de web server geïnstalleerd te worden.

Architectuur
Voor communicatie tussen een Desktop applicatie en remote DLL’s kan gebruikt gemaakt worden van Web Services of .NET Remoting. Het grote voordeel van Web Services is dat het hiermee mogelijk is om verschillende apparaten via het Internet met elkaar te laten communiceren. Indien een applicatie ook buiten het RIVM bereikbaar moet zijn, is dit de ideale oplossing.

Door programmacode in meerdere assemblies te verdelen die elk een eigen taak hebben (User Interface, de business rules en datalogica), wordt een n-tier structuur toegepast. Dit vraagt echter per laag een andere manier van programmeren. De .NET ontwikkelaar zal dus op meerdere markten getraind moeten zijn. Indien er meerdere ontwikkelaars werken aan een project kan een bepaalde rolverdeling toegepast worden, zodat de kennis van de ontwikkelaar optimaal benut wordt. Een ander groot voordeel van het gebruik van een n-tier structuur is dat de applicatie schaalbaar en onderhoudbaar wordt.

Keuze ontwikkeltaal
Voor het bouwen van de User Interface voor Desktop applicaties is VB.NET (voor een VB6 ontwikkelaar) het meest geschikt. De voorkeur van een C of een Java programmeur zal waarschijnlijk uitgaan naar C#. Overigens is dit slechts een persoonlijke voorkeur van de ontwikkelaar. Beide ontwikkeltalen onderscheiden zich slechts door de syntax waarin de code geschreven wordt en bij beide kan gebruik gemaakt worden van Windows Forms. Voor het bouwen van de User Interface van Web applicaties is ASP.NET het meest voor de hand liggend. De User Interface wordt dan gebouwd met behulp van Web Forms. Als programmeertaal kan hier ook voor VB.NET gekozen worden. Zowel Windows Forms als Web Forms bevatten veel controls waarmee het mogelijk is om rijke User Interfaces te bouwen.

Visual Basic.NET
Visual Basic .NET (VB.NET) zal een hoop herkenning geven voor de ervaren VB ontwikkelaars. Er zijn echter ook een aantal grote veranderingen doorgevoerd. De meest belangrijke is dat VB.NET compleet object georiënteerd is geworden. Het grote voordeel hiervan is dat applicaties beter schaalbaar en meer onderhoudbaar worden. Dit heeft niet alleen invloed op het schrijven van code, maar ook op keuzes voor de structuur van de applicatie. Verder zijn er een aantal wijzigingen doorgevoerd in de datatypes en de declaratie hiervan. Nieuw is onder andere de foutafhandeling. Voor de conversie van VB6 applicaties naar VB.NET is een wizard aanwezig in VS.NET. Het nut van de conversie zal echter per project bekeken moeten worden, omdat er vaak code overblijft die niet te converteren is door de wizard.

Tot slot bevat het Framework heel veel functionaliteit in de libraries. Het is dan ook belangrijk om eerst in het Framework te kijken of er functies aanwezig zijn, alvorens zelf iets te gaan programmeren.

Active Server Pages.NET
ASP.NET is een ontwikkeltaal voor het bouwen van Web applicaties en Web Services en is de opvolger van ASP. De code van ASP.NET wordt op een web server uitgevoerd waarna er een HTML pagina aan de gebruiker wordt getoond. ASP.NET biedt ten opzichte van ASP een groot aantal Web controls. Hiermee kan een rijkere User Interface gemaakt worden.

Bij het ontwikkelen van code voor een ASP.NET applicatie kunnen dezelfde .NET ontwikkeltalen gebruikt worden als bij Desktop applicaties. Dit kunnen ook niet Microsoft .NET talen zoals Delphi of Fortran zijn. Dit is een groot voordeel, want een ontwikkelaar hoeft geen nieuwe taal te leren om Web applicaties te bouwen. De code van een ASP.NET pagina wordt gescheiden opgeslagen in een zogenaamde Code-Behind file. De toevoeging van deze Code-Behind files zorgt voor een duidelijkere scheiding tussen lay-out en code.

Hardware en Software

De systeemeisen aan ontwikkel PC's zijn voor Visual Studio .NET toegenomen. Verder blijkt uit praktijkervaring dat pas vanaf Windows 2000 echt goed met Visual Studio .NET ontwikkeld kan worden. Op dit moment wordt bij het MNP echter nog gebruik gemaakt van Windows NT4.

Om een .NET applicatie te kunnen gebruiken is het .NET Framework nodig. Bij Desktop applicaties dient bij de gebruiker het .NET Framework geïnstalleerd te worden. Bij Web applicaties is het voldoende om het .NET Framework te installeren op de web server. Een Web applicatie heeft dus als groot voordeel dat het .NET Framework niet gedistribueerd hoeft te worden naar alle gebruikers. Het .NET Framework is overigens gratis te downloaden van de Microsoft website. Web applicaties worden ondersteund door alle browsers, maar werken het beste met Microsoft Internet Explorer 6.

Conclusies

Microsoft .NET is in de ICT wereld meer dan geaccepteerd. Niet alleen Microsoft producten zoals Windows, Office en SQL Server sluiten aan op de .NET technologie, maar ook Unix en Oracle zullen (binnenkort) deze nieuwe technologie ondersteunen. Hierdoor sluit de .NET technologie goed aan op de MNP infrastructuur en op de legacy systemen.

Microsoft .NET biedt veel mogelijkheden. Doordat .NET veel ontwikkeltalen ondersteunt en het Framework zeer uitgebreid is, heeft het veel toegevoegde waarde voor het MNP. Ook is het vrij eenvoudig om een rijke User Interface voor zowel Desktop als Web applicaties te maken.

Met de komst van assemblies is niet alleen de DLL-hell opgelost, maar is ook de distributie een stuk eenvoudiger geworden. Zowel assemblies als de distributie bieden grote voordelen bij het beheren en uitrollen van applicaties.

Voor nieuwbouw applicaties is .NET zeker één van de betere oplossingen. Bij bestaande applicaties blijft het belangrijk een afweging te maken tussen de voordelen van .NET en de investering die het kost om de applicatie om te zetten naar een .NET applicatie. Als vuistregel kan genomen worden dat bij grote applicaties de conversie net zoveel tijd in beslag zal nemen als het opnieuw bouwen van de applicatie. Op het moment dat de wens er is om een applicatie een Web User Interface te geven is een goed moment om de applicatie om te zetten in een .NET applicatie.

Om optimaal te kunnen profiteren van de voordelen van .NET zal echter wel geïnvesteerd moeten worden in het opbouwen van kennis. Deze kennis zal zich zeker terug verdienen in betere, schaalbare en onderhoudbare applicaties.

Inhoud

1. Inleiding

Microsoft .NET (spreek uit: dot NET) is een nieuwe technologie voor het koppelen van informatie, mensen, systemen en devices. Niet voor niets is de slogan ‘Microsoft .NET – Connected Software’. Dit klinkt heel breed en dat is het ook. Daarom wordt in deze Reference Guide eerst aandacht besteed aan de .NET technologie zelf, waarna er dieper op verschillende aspecten van deze technologie wordt ingegaan. Microsofts visie is dat iedereen op een willekeurig apparaat over zijn of haar informatie kan beschikken. Doordat de .NET technologie zo uitgebreid is, kan in deze Reference Guide niet op alles ingegaan worden.

Binnen het MNP wordt het standaard om alle nieuwe applicaties Web enabled te maken. Mede hierom wordt er in deze Reference Guide vooral aandacht gegeven aan Web applicaties en de benodigde architectuur.

1.1 Doel van de Reference Guide

Zoals gezegd is het toekomstperspectief van het MNP dat alle nieuwe applicaties ontsluiting moeten hebben via het Web. Op dit moment zijn er twee stromingen in nieuwe technologieën waarmee dat gerealiseerd kan worden, te weten J2EE (Java) en .NET (Microsoft). Doordat een groot aantal applicaties van het voormalige LAE ontwikkeld zijn met VB(A) wordt in dit document de nieuwe technologie van Microsoft onder de loep genomen.

Het is niet de bedoeling om de nieuwe technologieën van Java en Microsoft met elkaar te vergelijken, of om aan te tonen welke beter zou zijn. Het doel van deze Reference Guide is echter om het nut en de toepasbaarheid van .NET te beschrijven. Doordat veel aandacht wordt besteed aan de uitgebreide .NET technologie is de Reference Guide tevens een handreiking voor ontwikkelaars die gaan ontwikkelen met VS.NET.

1.2 Inhoud van de Reference Guide

De ‘Microsoft .NET Reference Guide’ bevat een globale omschrijving van de .NET technologie en voorkeurswerkwijzen voor ontwikkelaars over inzet en gebruik van de beschikbare ontwikkeltools binnen de .NET technologie.

Sinds medio 2001 is er een opvolger van het veel gebruikte Visual Studio 6 waarin Visual Basic 6 (VB6) is opgenomen. Deze opvolger is Visual Studio.NET met daarin de opvolger van VB6 te weten Visual Basic.NET. De Integrated Development Environment (IDE) van VS.NET is dé ontwikkelomgeving voor alle ontwikkeltalen binnen de .NET technologie.

Zoals reeds gezegd is de .NET technologie erg uitgebreid. In deze Reference Guide wordt een algemene indruk gegeven van wat de .NET technologie omvat, en hoe deze gebruikt kan worden binnen het MNP. Door het vroegere LAE werd vooral gewerkt met VB6 en Visual Basic for Applications (VBA) in Office 97 als ontwikkelomgeving. Om een goed vergelijk te kunnen maken zal in deze Reference Guide dieper ingegaan worden op VB.NET en geen aandacht besteed worden aan de nieuwe ontwikkeltaal C# (spreek uit: Sie Sjarp).

Omdat MNP applicaties ontsluiting krijgen via het Web wordt ook dieper ingegaan op ASP.NET. ASP.NET is de opvolger van ASP en waarmee Web applicaties en Web Services gebouwd kunnen worden.

De meeste huidige VB applicaties van het MNP maken gebruik van een database. De communicatie met de database verloopt via de door LAE-RIS ontwikkelde datakoppeling (RIS-datalaag) of rechtstreeks via de Microsoft standaard ActiveX Data Objects (ADO). Reeds eerder is besloten om niet verder gebruik te maken van de RIS-datalaag. De opvolger van ADO is ADO.NET. In de Reference Guide zal dieper ingegaan worden op de verschillen tussen ADO en ADO.NET en welke voordelen ADO.NET biedt.

Verder besteedt deze Reference Guide aandacht aan XML aangezien dit door Microsoft gezien wordt als de taal voor het ontsluiten van gegevens uit databases waarmee het mogelijk is om gegevens tussen verschillende componenten / applicaties uit te wisselen.

Tot slot wordt er beschreven aan welke eisen moet worden voldaan om .NET applicaties te kunnen ontwikkelen en te kunnen gebruiken.

1.3 Leeswijzer

In hoofdstuk 2 wordt een algemene beschrijving gegeven van de .NET technologie. In de daarop volgende hoofdstukken wordt er op bepaalde onderwerpen dieper ingegaan.

In hoofdstuk 3 wordt een beschrijving gegeven van de ontwikkeltalen VB.NET, ASP.NET, XML en ADO.NET.

In hoofdstuk 4 wordt ingegaan op de nieuwe architectuur die mogelijk is bij grote .NET applicaties. Achtereenvolgens komt de infrastructuur, authenticatie, user interface, Web Services, de business laag en de data laag ter sprake. Per laag zal ook de te gebruiken ontwikkeltaal en de toepassing hiervan beschreven worden.

Hoofdstuk 5 gaat dieper in op de User Interface en de verschillen tussen Desktop en Web applicaties.

In het laatste hoofdstuk is een beschrijving te vinden voor de inrichting van PC’s voor het ontwikkelen, gebruiken en hosten van .NET applicaties.

2. Algemeen

2.1 MNP en .NET

De vraag of het MNP over moet gaan naar .NET kan simpelweg beantwoord worden met ‘ja’. Met .NET heeft Microsoft een nieuwe weg ingeslagen voor het maken van applicaties, het ontsluiten van data en het koppelen van verschillende systemen. Hiervoor is Microsoft druk bezig alle software die ze op de markt brengen geschikt te maken voor .NET. Dit betekent dat nieuwe versies van Windows, SQL Server, Office etc. allemaal .NET zullen gaan ondersteunen, of er zelfs primair gebruik van gaan maken.

Dit betekent echter niet dat het MNP alleen nog maar .NET moet gaan gebruiken. Per applicatie zal een afweging gemaakt moeten worden of het inzetten van de .NET technologie nuttig is.

Om beter inzicht te krijgen waarom het MNP ook .NET ‘moet’ gebruiken zal in dit hoofdstuk ingegaan worden op wat de technologie precies inhoudt. In de daarop volgende hoofdstukken zal gedetailleerder naar de techniek gekeken worden. Op basis van deze informatie wordt in de conclusie toegelicht wat het nut en de toepasbaarheid van .NET is voor het MNP.

2.2 Architectuur

Voordat met de bouw van een applicatie begonnen kan worden, moet eerst nagedacht worden over de architectuur van de applicatie. De architectuur behelst de technische opzet van een applicatie en keuzes hierover hebben grote invloed op het succes van de applicatie. De keuze van de architectuur van een applicatie is onder andere afhankelijk van:

- de (gewenste minimale) responstijd;
- de beschikbare capaciteit;
- het gewenste niveau van beveiliging;
- de gewenste betrouwbaarheid;
- gewenste integratie met andere applicaties;
- implementatie mogelijkheden.

De meest gebruikte architecturen voor applicaties met een onderliggende database is de 2-tier, 3-tier en de n-tier structuur. De bestaande VB6 applicaties binnen het MNP zijn tot nu toe hoofdzakelijk met een 2-tier structuur opgebouwd.

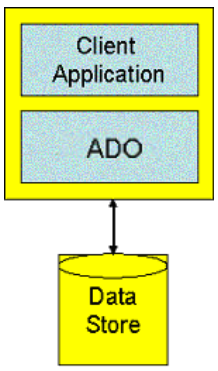


Fig. 1: 2-tier applicaties bij het MNP

Bij de 2-tier structuur beschikken alle clients over de applicatie, die met behulp van ADO connectie maakt met een database. Hierbij wordt bij voorkeur gebruik gemaakt van disconnected recordsets. Dit zorgt voor minder netwerkverkeer en meer controle over de data.
Bij een 3-tier applicatie is een aparte laag gecreëerd waarin de database met behulp van ADO wordt aangesproken. Voordeel hiervan is dat deze laag ook op een aparte machine dichtbij de database kan worden geplaatst. Dit wordt meestal gedaan vanwege performance eisen.
Bij een n-tier structuur is de applicatie opgedeeld in nog meer lagen. Meer hierover is te lezen in hoofdstuk 4.

2.3 .NET technologie

Zoals gezegd is .NET slechts een technologie en kan .NET dus niet geïnstalleerd worden. De visie die Microsoft met de .NET technologie heeft, is om applicaties met behulp van een n-tier structuur te bouwen. De communicatie tussen de lagen geschiedt met behulp van XML. Hierdoor ontstaan schaalbare, onderhoudbare en vooral database onafhankelijke applicaties. Hierdoor lijkt het erop dat er binnen .NET geen keuze meer gemaakt hoeft te worden voor de te gebruiken architectuur. Niets is minder waar, .NET ondersteunt nog steeds de gebruikelijke architecturen en vereenvoudigt deze zelfs door het uitgebreide .NET Framework. Het blijft dus van belang om een goede architectuur te kiezen.

Om de .NET technologie meer te visualiseren wordt hieronder een globale weergave getoond, zoals Microsoft haar nieuwe technologie ziet.

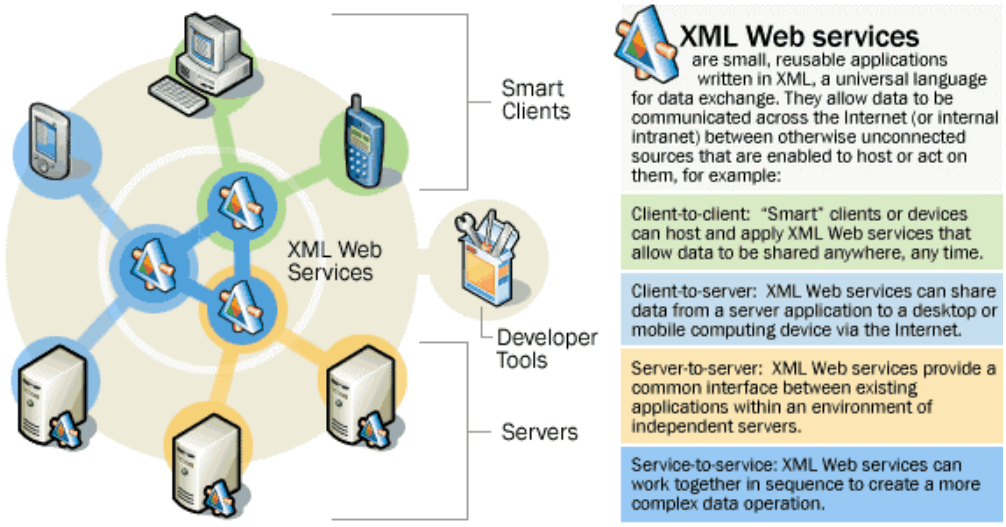


Fig. 2: .NET Technologie

2.3.1 Smart Clients

Onder Smart Clients worden de PC's, laptops, telefoon, handheld computers (PDA's), Tablet PC's en zelfs de Microsoft XBox game console verstaan. Deze apparaten worden 'smart' genoemd, omdat ze in staat zijn Web Services te benaderen. Hierdoor kunnen ze beschikken over door de gebruiker gewenste data, onafhankelijk van welk apparaat gebruikt wordt en onafhankelijk van waar de data is opgeslagen.

Fig. 3: Smart Clients

Om de .NET technologie te kunnen gebruiken moeten de apparaten beschikken over de laatste versie van het Windows besturingssysteem. Dit zijn Windows XP, Pocket PC, Windows CE.NET, Windows XP Embedded, Windows Powered Smartphone 2002 en Windows XP Tablet PC Edition. Het is echter ook mogelijk om bijvoorbeeld .NET applicaties te draaien onder Windows 2000. Hiervoor moeten echter extra Windows componenten geïnstalleerd worden (zie verder paragraaf 6.2).

2.3.2 Web Services

Voordat het mogelijk is om Web Services te beschrijven moet eerst het begrip XML uitgelegd worden. XML staat voor eXtended Markup Language en is een universele taal om gegevens te beschrijven. Meestal zijn dit gegevens uit een database, maar met XML is het ook mogelijk om een database te vervangen door XML documenten. XML wordt tevens gebruikt om de interface van een Web Service te beschrijven.
Zie verder paragraaf 3.4 voor een uitgebreidere beschrijving van XML.

Web Services zorgen voor de communicatie tussen de verschillende clients. Dit is onafhankelijk van de operating systems en onafhankelijk van de ontwikkeltaal waarin de applicaties en de Web Services gemaakt zijn. Web Services kunnen onafhankelijk van elkaar werken, maar zijn ook zeer geschikt om samen een taak uit te voeren. Een Web Service is dus de intermediair voor data tussen verschillende devices en verschillende servers, zoals te zien is in figuur 2 (XML Web Services : de spin in het web).

Web Services voldoen aan een aantal standaards, te weten: Simple Object Access Protocol (SOAP), XML standaards en Universal Description, Discovery, and Integration (UDDI). Deze standaards zijn beschreven en worden onderhouden door onder andere het World Wide Web Consortium (W3C). Deze begrippen worden in paragraaf 4.4.1 toegelicht.

2.3.3 Servers

Windows heeft een groot scala aan servers die gebruikt kunnen worden voor het hosten van Web Services en .NET applicaties. De Microsoft .NET Enterprise Servers, Windows 2000 Server family, en de aankomende Windows .NET Server family zijn uitermate geschikt hiervoor.

Natuurlijk kunnen ook andere servers gebruikt worden. Er moet dan wel goed bekeken worden of deze beveiliging en XML ondersteuning bieden die voor .NET applicaties benodigd zijn. De meeste nieuwe versies van servers van de concurrenten van Microsoft zorgen er echter wel voor dat ze goed aansluiten op de nieuwe .NET technologie.

Een voorbeeld van servers uit de .NET Enterprise Server family zijn:
- Microsoft Application Center 2000 voor het gebruik en beheer van Web applicaties.

- Microsoft Mobile Information 2001 Server voor ondersteuning van applicaties op mobile devices zoals telefoons.
- Microsoft SharePoint Portal Server 2001 voor het zoeken, delen en publiceren van bedrijfsinformatie (de tegenhanger van Lotus Notes).
- Microsoft SQL Server 2000 voor het bewaren, ophalen en analyseren van gestructureerde XML data.

2.3.4 Developer Tools

Met behulp van Visual Studio.NET (VS.NET) en .NET Framework is het mogelijk om Web Services en .NET applicaties te maken. VS.NET omvat een uniforme omgeving voor het ontwikkelen in alle beschikbare .NET talen. Hierdoor is het onder andere mogelijk om de gehele applicatie over verschillende ontwikkeltalen te debuggen. VS.NET is er in drie verschillende uitvoeringen, te weten: Enterprise Architect, Enterprise Developer en Professional. De belangrijkste verschillen staan in de volgende tabel weergegeven.

Tabel. 1: Belangrijkste verschillen tussen versies van Visual Studio .NET

Opties	Enterprise Architect	Enterprise Developer	Professional
SQL Server 2000	V	V	
Visio based database modeling	V		
Visual Sourcesafe	V	V	
Application Center Test	V	V	
Visio based UML modelling	V		
Template project type	V		
Templates en framework	V	V	
Reference applications	V	V	
Visual Studio Analyser	V	V	

Verder bevat zowel de Enterprise Architect versie als de Enterprise Developer versie volledige versies van Windows 2000 Standard Server, SQL Server 2000, Microsoft Commerce Server, Microsoft Host Integration Server, Microsoft Exchange Server en Microsoft BizTalk Server. Deze producten hebben alle een zogenaamde 'Development en Test license'.
In hoofdstuk 3 wordt verder ingegaan op de diverse ontwikkeltalen in VS.NET.

2.4 .NET Framework

Het .NET Framework is het gestandaardiseerde programmeermodel van de Microsoft .NET technologie. Het .NET Framework is geschikt voor het bouwen van Web applicaties, Smart Client Applicaties en Web Services.
Het .NET Framework bestaat uit twee delen, de Common Language Runtime en een set Class Libraries waaronder ASP.NET, Microsoft Forms en ADO.NET.

2.4.1 Common Language Runtime

De Common Language Runtime (CLR) bevat de compilers en tools voor het uitvoeren van .NET applicaties. Tijdens het compileren van een applicatie wordt de code omgezet in een CLR Portable Executable (PE). Een gecompileerd .NET project (een EXE of een DLL bestand) wordt ook wel een assembly genoemd. Een .NET applicatie bestaat dus uit minimaal één assembly. Een assembly bestaat uit een deel dat MetaData bevat en een deel dat de werkelijke code bevat. Deze code wordt voor elke .NET taal geconverteerd naar de Microsoft Intermediate Language (MSIL).

De MetaData beschrijft de code in een neutrale taal en bevat de identiteit (naam, versie etc.), de assemblies waarvan deze assembly afhankelijk is en de beveiliging van de assembly. En verder bevat de MetaData informatie over de gebruikte base classes, interfaces en de methods, fields, properties en events van de assembly.

.NET applicaties worden gestart met behulp van de zogenaamde Just-In-Time (JIT) compilation. Dit wil zeggen dat, zodra de applicatie door een gebruiker wordt gestart, de MSIL code in de PE file wordt gecompileerd tot de uiteindelijke native code die door de computer in het geheugen wordt uitgevoerd. Doordat de CLR op de client alleen MSIL code hoeft te compileren maakt het niet uit in welke ontwikkeltaal de applicatie geschreven is. Hierdoor kunnen dus ook binnen een assembly meerdere talen gebruikt worden. Per classe

moet alles in één taal geschreven worden, maar verschillende forms, classes of modules kunnen wel geschreven zijn in andere taal.

Om .NET applicaties te kunnen uitvoeren is de CLR nodig. Dit betekent dat het .NET Framework op de client geïnstalleerd moet worden. Dit geldt echter niet voor Web applicaties. De code van een Web applicatie draait namelijk op een web server die HTML genereert en dit aanbiedt in de browser van de client. Het is dus wel noodzakelijk om de CLR op de web server te installeren, maar niet op alle clients individueel. Dit is een groot voordeel ten opzichte van Desktop applicaties, waarbij de CLR dus wel nodig is op alle clients. Een uitgeklede versie van het .NET Framework, dat geschikt is om .NET applicaties te kunnen uitvoeren, is te vinden op:
http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/829/msdncompositedoc.xml

Doordat de assembly zelf beschikt over MetaData, dat informatie bevat over de code, de interface etc. van de assembly, is daarmee ook de zogenaamde DLL-hell opgelost. Het is namelijk niet meer nodig om de assembly te registreren (in de registry), aangezien de meest recente benodigde informatie aanwezig is in de assembly zelf.
De DLL-hell wordt bovendien opgelost door het feit dat er meerdere versies van de assembly naast elkaar kunnen staan. In de applicatie kan aangegeven worden of een globale assembly of een assembly speciaal voor de applicatie gebruikt moet worden. Een applicatie zal altijd eerst in de assembly directory van de applicatie kijken. Is er geen assembly aanwezig dan wordt in de Global Assembly Cache gekeken (zie ***Fig. 4***). De Global Assembly Cache is de centrale locatie op een PC waarin alle assemblies staan die door meerdere applicaties gebruikt mogen worden. Het is daarom ook alleen aan te raden om gebruik te maken van de Global Assembly Cache indien de assembly door meerdere applicaties gebruikt moet worden. In elk andere situatie is het beter om de assembly in de directory van de applicatie te plaatsen. Dit levert namelijk voordeel op bij een setup. Een assembly kan alleen in de Global Assembly Cache geplaatst worden met behulp van de dos-tool ‘gacutil.exe’ uit het Framework. Met behulp van het commando ‘gacutil.exe /?’ kan help opgevraagd worden over de exacte werking.

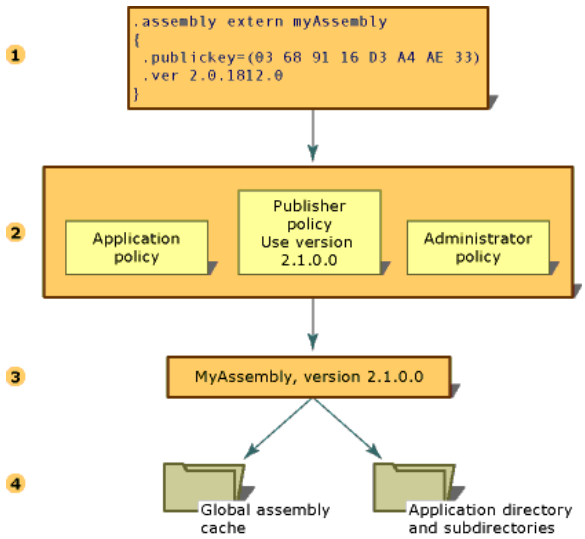


Fig. 4: Globale en applicatie assemblies

Zowel in de Global Assembly Cache als in de Application directory c.q. subdirectory kunnen van een assembly meerdere versies staan. In elke assembly is, net zoals in VB6, een

versienummer opgenomen in de vorm van : *major.minor.build*. Op het moment dat een referentie naar een assembly wordt gelegd kan aangegeven worden hoe gevoelig deze referentie moet zijn. Zo kan bijvoorbeeld opgegeven worden dat de referentie wel geüpdate kan worden bij het ophogen van een nieuw build nummer, maar dat de referentie geen gebruik mag maken van een nieuwe assembly met een hoger minor nummer. (Dit is de standaard instelling binnen VS.NET).

Voorbeeld:
Een Applicatie (EXE) maakt gebruik van een DLL met versienr : 1.2.468
De DLL bevat een kleine bug en wordt opnieuw gedistribueerd met versienr : 1.2.469
De applicatie maakt nu gebruik van de nieuwe DLL met versienr 1.2.469
De DLL krijgt nieuwe functionaliteit en wordt gedistribueerd met versienr : 1.3.470
De applicatie zal nu gebruik blijven maken van de bestaande DLL met versienr 1.2.469
Overigens zijn alle drie de DLL's in dit voorbeeld aanwezig in Global Assembly Cache.

In bovenstaand voorbeeld wordt aangegeven dat er drie DLL's in de Global Assembly Cache staan. Voorheen was dit niet mogelijk, aangezien een nieuwe versie van de DLL dezelfde naam heeft. In .NET moet een DLL echter voldoen aan zogenaamde strong name naamgeving. Dit betekend dat de naam moet zijn opgebouwd uit :
- bestandsnaam zonder extensie (MyDll);
- versienummer (1.2.648);
- cultuur (NL);
- strong name info, bestaat onder andere uit Public Key Token.

Een Public Key Token is een unieke sleutel die elke ontwikkelaar kan aanmaken om zijn assemblies mee te 'ondertekenen'. Dit zorgt ervoor dat elke DLL die ontwikkelaars maken uniek van elkaar zijn. Hiermee is het dus ook mogelijk om meerdere versies op één PC te hebben staan.

De CLR zorgt ervoor dat .NET applicaties werken op alle platforms waarop de CLR geïnstalleerd kan worden. Microsoft garandeert dat de CLR werkt vanaf Windows 98, eventueel met de benodigde service packs. (zie ook paragraaf 6.1)

2.4.2 Class Libraries

De Forms zoals deze gebruikelijk waren in VB6 zijn niet meer te gebruiken in .NET. Hiervoor dient de Form class library van .NET gebruikt te worden. Afgezien van het verdwijnen van een paar weinig gebruikte controls, is dit alleen maar een voordeel. De Forms library bevat veel controls, waaronder bijvoorbeeld de common dialogs. Ook de manier van omgaan met forms is veranderd, doordat forms en controls nu als objecten gezien worden. De veranderingen zijn uitgediept in paragraaf 5.3.1.

De class library ASP.NET is zeer geschikt voor het ontwikkelen van Web applicaties en het bouwen van Web Services. De class library ASP.NET bevat onder andere een groot aantal controls die geschikt zijn voor het gebruik in browsers. Het aantal controls is kleiner dan die in de Forms class library, maar bieden toch een ruime keus bij het bouwen van Web User Interfaces. Voor het programmeren van code in ASP.NET kan gekozen worden uit de standaard aanwezige ontwikkeltalen VB.NET en C#. Daarnaast komen er steeds meer ontwikkeltalen zoals Fortran.NET en Delphi.NET die gebruikt kunnen worden voor ASP.NET web applicaties. In paragraaf 3.3 wordt ASP.NET verder uitgewerkt.

Verder is de class library ADO.NET beschikbaar voor het ontsluiten van data uit databases, text files, XML files etc. Belangrijk bij ADO.NET is dat het uitgaat van zogenaamde 'light-

touch'clients. Dit wil zeggen dat de client connectie maakt met de database, de data ophaalt en de connectie weer verbreekt (vergelijkbaar met disconnected recordsets). Dit is reeds zeer gebruikelijk bij het bouwen van n-tier applicaties.
Het kan echter voordelen hebben om juist de 'heavy-touch' benadering te gebruiken. De client houdt de connectie open, zolang de client de gegevens nodig heeft. Het is dan ook belangrijk om te weten dat ADO.NET alleen forward-only, read-only cursors ondersteunt voor het lezen van data. Stateful server-side cursors worden niet meer ondersteund om korte connecties naar de database aan te moedigen. Meer over ADO.NET is te vinden in paragraaf 3.5.

Tot slot bevat het Framework nog veel meer class libraries. Hierbij kan gedacht worden aan de Common Dialog schermen, File System Object etc. Deze classes moesten in VB6 nog veelal met add-ins of API's geprogrammeerd worden. Het is daarom ook belangrijk om eerst goed te zoeken wat het Framework biedt alvorens zelf code te schrijven. Met behulp van de MSDN library kan goed gezocht worden binnen het Framework.

3. Ontwikkeltalen

3.1 Taalkeuze

Visual Studio.NET is een ontwikkelomgeving (IDE) voor alle door Microsoft ontwikkelde .NET programmeertalen voor het maken van .NET applicaties. Ook de concurrenten van Microsoft zorgen dat ze aansluiten op het .NET Framework. Zo bestaan er momenteel al circa 30 .NET talen en binnenkort komen er nog meer .NET talen uit zoals: Delphi.NET en Cobol.NET.

Gezien de VB6 ervaring is voor het bouwen van de User Interface voor Desktop applicaties VB.NET het meest geschikt. Met VB.NET kan gebruik gemaakt worden van Windows Forms voor de User Interface. VB.NET is qua syntax niet veel veranderd ten opzichte van VB6. Dit is dus een relatief gemakkelijke overstap. Voor het bouwen van de User Interface van Web applicaties dient ASP.NET gebruikt te worden. De User Interface wordt dan gebouwd met behulp van Web Forms. Als programmeertaal kan hier ook voor VB.NET gekozen worden.

In dit hoofdstuk wordt ASP.NET overigens ook behandeld ondanks dat het eigenlijk geen taal is, maar een Class library. Dit is enigszins verwarrend, waar in dit document dan ook gesproken wordt over ASP.NET als programmeertaal, zal dit gezien moeten worden als de syntax en de controls van de ASP.NET Class inclusief de programmeertaal (bijv. VB.NET) die gebruikt wordt.

3.2 VB.NET

3.2.1 Verschillen tussen VB6 en VB.NET

Binnen VS.NET is het mogelijk om verschillende ontwikkeltalen door elkaar te gebruiken. Alle talen worden zoals reeds eerder genoemd gecompileerd tot een Intermediate Language. Door deze integratie van verschillende talen zullen alle talen dezelfde eigenschappen moeten hebben. De grootste verschillen zijn dan ook te vinden in de variabele types en in het feit dat VB.NET nu volledig object georiënteerd is.

Types

Het eerste verschil (dat niet echt te zien is), is het feit dat VB.NET geschikt is om 64-bit applicaties te bouwen. Dit betekent dat de waarde van een aantal bekende types veranderd is. Zo bevat een Integer in VB6 16 bits, en in VB.NET 32-bits! Bovendien zijn er een aantal types in VB.NET verdwenen en een aantal andere types geïntroduceerd. Belangrijk is ook te weten dat alle variabelen geconverteerd moeten worden in het juiste type. Een variabele van het type Integer is dus niet zomaar meer in een variabele van het type Long te stoppen.

Tabel. 2: Integer types in VB.NET

Integer Size	VB6 type & type char	VB.NET type & type char	CLR type
8 bits, signed	(none)	(none)	System.SByte
16 bits, signed	Integer (%)	Short (none)	System.Int16
32 bits, signed	Long (&)	Integer (%)	System.Int32
64 bits, signed	(none)	Long (&)	System.Int64

Toegevoegd zijn de types Char, Short, en Decimal. Dit voor volledige compatibiliteit met C# en C++ projecten.

Tabel. 3: VB.NET primitive types

Name	Object	Description	Default Value
Byte	System.Byte	1 byte unsigned integer	0
Short	System.Int16	2 byte signed integer	0
Integer	System.Int32	4 byte signed integer	0
Long	System.Int64	8 byte signed integer	0
Single	System.Single	4 byte floating point	0
Double	System.Double	8 byte floating point	0
Decimal	System.Decimal	16 byte decimal	0D
Boolean	System.Boolean	True or False (1 or 0)	False
Date	System.DateTime	Date time stamp	#01/01/0001 12:00:00 AM#
Char	System.Char	Single unicode character	ChrW(0)
String	System.String	Multiple unicode characters	null reference

Variant
Een andere verandering in de declaratie van variabelen is dat het type ‘Variant’ is verdwenen. Deze is vervangen door ‘Object’ (VB.NET is volledig object georiënteerd).

Boolean
Een klein verschil, maar wellicht met grote gevolgen is het feit dat de ‘True’ waarde van een boolean van –1 naar 1 is gegaan.

Declaratie
Ook in de declaratie van variabelen is het een en ander veranderd. Zo was het in VB6 ‘not done’ om de declaratie `Dim a, b, c, d, e As Integer` te gebruiken. Immers alleen e is in dit geval van het type Integer, alle andere variabelen zijn van het type Variant. In VB.NET is dit aangepast. Bij de hiervoor genoemde declaratie zullen alle variabelen in VB.NET van het type Integer zijn. Ook is het mogelijk om in VB.NET een variabele tijdens de declaratie gelijk een waarde mee te geven, of de waarde van een andere variabele.

```
Dim myInt As Integer = 25
Dim otherInt As Integer = myInt * 100
```

Een andere niet opvallende verandering is dat de plaats van declaratie van belang is bij het gebruik van die variabele. In VB6 was een op een willekeurige plaats gedeclareerde variabele binnen de scope van de functie (of module) te gebruiken. Bij VB.NET is het zo dat een variabele die binnen een set van statements dat eindigt met End.., Loop of Next niet buiten deze statements te gebruiken zijn.

```
Dim teller1 As Integer
  For teller1 = 0 To 10
    Dim teller2 As Long
    teller2 = i * 3
  Next teller1
  MsgBox(teller2)
```

In VB.NET zal het laatste statement de volgende fout genereren: ‘The variable ‘teller2’ has not been declared’. Dit komt doordat de variabele teller2 buiten het For Next block wordt aangesproken.

Ook bij de enumeraties is een wijziging doorgevoerd. In VB6 is een enumeratie altijd een Long en kan een type verder niet gedeclareerd worden. In VB.NET kan een enumeratie met de volgende types gedeclareerd worden: Byte, Short, Integer of Long. Standaard is een enumeratie van het type Integer (de oude Long van VB6).
De volgende declaraties zijn dus hetzelfde in VB.NET:

```
Public Enum KleurEnum as Integer
      rood = 0
      groen = 1
      blauw = 2
      oranje = 3
End Enum

-en-

Public Enum KleurEnum
      rood
      groen
      blauw
      oranje
End Enum
```

Option Explicit / Option strict
Voor het afdwingen van het declareren van variabelen bestaat in VB6 Option Explicit. Over het feit dat deze altijd aan moet staan is bijna iedereen het wel eens. In VB.NET staat deze optie default dan ook aan. Een nieuwe optie is Option Strict. Deze optie zorgt ervoor dat bijvoorbeeld de waarde van een Integer variabele niet zomaar in een String variabele gestopt mag worden. Hiervoor dient een type conversie uitgevoerd te worden. Wel mag bij numerieke waarden een zogenaamde ‘widening convertion’ uitgevoerd worden. Dit wil zeggen dat de nieuwe variabele groter is dan de oorspronkelijke variabele. Een waarde uit een Integer variabele mag dus wel in een Long variabele gestopt worden. Let hierbij wel op dat een conversie van een Integer naar bijvoorbeeld een Single of een Double geen precisie (decimalen) bevat.

Object Georiënteerd (OO)
De grote merkbare verschillen tussen VB6 en VB.NET zijn vooral dat VB.NET volledig object georiënteerd is. Voor ontwikkelaars die gewend zijn om OO applicaties in VB6 te maken, zal dit een grote vooruitgang betekenen. Voor ontwikkelaars die dit niet gewend zijn, zal het een behoorlijke omschakeling vragen. Voor het OO programmeren is nu (eindelijk) ook de beschikking over de mogelijkheid van inheritance, interfaces en overloading. Dit wordt hierna toegelicht.

Het grote voordeel van OO is dat de programmeercode ingedeeld wordt in objecten die gerelateerd zijn aan het onderwerp waarover geprogrammeerd wordt. Om dit te concretiseren wordt vaak als voorbeeld een programma voor auto's genomen. In deze applicatie zal een object van het type auto gemaakt worden, welke een collectie van wielen zal hebben. De collectie van wielen zal vervolgens weer uit wiel objecten bestaan. Het hoeft nu voor een ontwikkelaar niet meer moeilijk te zijn om te bedenken waar hij de kleur van de auto en waar hij het merk van de autobanden moet vinden (respectievelijk in het auto object en in het wiel object). Deze manier van programmeren zorgt voor schaalbare en beter onderhoudbare applicaties.

Het verschil in object georiënteerdheid is allereerst te merken in de manier van benaderen van forms. Deze worden nu als object gezien en dienen dan ook eerst geïnstantieerd te worden alvorens ze geopend kunnen worden. Zodra de variabele die het form object bevat de scope verliest, wordt ook het form ‘geunload’.

```
Public Sub Main()
    Dim hoofdScherm as New MainForm()

 hoofdScherm.Show()

End Sub
```

In de eigenschappen van een project kan, zoals reeds bekend bij VB6, een object opgegeven worden dat de applicatie als eerst moet opstarten. Dit kan een formulier, of een module zijn. Voor het openen van het eerste form vanuit een module dient gebruik gemaakt te worden van het Application object. Dit om te voorkomen dat, zoals hiervoor genoemd, het form gelijk weer gesloten wordt, zodra de code in Sub Main is uitgevoerd.

```
Public Sub Main()
    Application.Run(New MainForm())
End Sub
```

Statement ‘Set’

Wat verder veranderd is, is dat het statement Set niet meer nodig is voor het instantieren van objecten. Doordat VB.NET nu volledig object georiënteerd is, snapt VB.NET dat het om een object gaat. Tijdens het schrijven van code zal VB.NET dan ook controleren of beide objecten links en rechts van het ‘=‘teken van hetzelfde type zijn.

Constructor

De constructor zorgt voor de instantiatie van een object. Een classe wordt geïnstantieerd met ‘New ‘(gelijk aan VB6).
Een uitbreiding ten opzichte van VB6 is dat er aan de constructor parameters meegegeven kunnen worden. Hierdoor is het mogelijk om een class initieel een bepaalde waarde te geven.

```
Public Class Persoon
  Private _naam As String
  Private _geboortedatum As Date

  Public Sub New(ByVal naam As String, _
                 ByVal geboortedatum As Date)
    _naam = naam
    _geboortedatum = geboortedatum
  End Sub

End Class
```

Deze class kan nu op de volgende manier aangeroepen worden:

```
Dim Geert as Persoon = _
    new Persoon("Geert Verspaij", 31/12/64)
```

Overloading

Overloading betekent dat één functie meerdere 'signatures' kan hebben (op meerdere manieren ingevuld kan worden). De signature wordt bepaald door het aantal parameters en de data-type van elke parameter. In de praktijk bestaat een implementatie uit één signature. Bij het gebruik van de functie bepaalt de compiler aan de hand van de 'signature' welke functie wordt uitgevoerd. Overloading kan ook toegepast worden op de constructor.
N.B. In het voorbeeld wordt de 'toString' functie gebruikt. Elk object heeft een toString functie. Elke class erft automatisch van de 'object' class. Vandaar dat de Class Persoon in het voorbeeld ook een toString functie heeft. (De functionaliteit Override wordt in de volgende paragraaf uitgelegd.)

```
Public Class Persoon
  Private _naam As String
  Private _geboortedatum As Date
  Private _aantalKinderen As Integer

  Public Sub New(ByVal naam As String, _
                 ByVal geboortedatum As Date, _
                 ByVal aantalKinderen As Integer)
    _naam = naam
    _geboortedatum = geboortedatum
    _aantalKinderen = aantalKinderen
  End Sub

  ' Aantal kinderen is default 0,
  ' indien deze parameter niet wordt meegegeven
  Public Sub New(ByVal naam As String, _
                 ByVal geboortedatum As Date)
    Me.new(naam, geboortedatum, 0)
  End Sub

  ' Override de toString functie
  Public Overrides Function toString() As String
    Return naam & " geboren op: " & geboortedatum
  End Function

End Class
```

Deze class kan nu op de volgende manier aangeroepen worden:

```
Dim geert as Persoon = _
    new Persoon("Geert Verspaij", 31/12/64)
Dim wim as Persoon = _
    new Persoon("Wim van der Maas", 10/04/59, 3)
```

Inheritance

Inheritance beschrijft de mogelijkheid om nieuwe classes te maken die afgeleid zijn van bestaande classes. De nieuwe class erft alle properties, methoden en events van de base class. Van hieruit kan de class uitgebreid worden met nieuwe properties en methoden. Implementaties van de base class kunnen ook overschreven worden.

```
Public Class Medewerker
  Inherits Persoon
  Private _teamnaam As String

  Public Sub New(ByVal naam As String, _
                 ByVal leeftijd As Date,_
                 ByVal teamnaam As String)
    ' Gebruik de constructor van de base-class
    MyBase.new(naam, leeftijd)
    _teamnaam = teamnaam
  End Sub

  ' Override de toString functie
  Public Overrides Function toString() As String
    Return MyBase.ToString & " van team " & teamnaam
  End Function

End Class

Public Class Consultant
  Inherits Persoon
  Private _bedrijfsnaam As String

  Public Sub New(ByVal naam As String, _
                 ByVal leeftijd As Date,_
                 ByVal bedrijfsnaam As String)
    ' Gebruik de constructor van de base-class
    MyBase.new(naam, leeftijd)
    _bedrijfsnaam = bedrijfsnaam
  End Sub

  ' Override de toString functie
  Public Overrides Function toString() As String
    Return MyBase.ToString & " van bedrijf " & bedrijfsnaam
  End Function

End Class
```

De volgende aanroepen zijn mogelijk:

```
Dim MarcDeB as Consultant = _
    new Consultant ("Marc de Bot", 12-4-1964, "VXCompany")
Dim MarkVdZ as Medewerker = _
    new Medewerker ("Mark van der Zee", 8-9-1975, "IMP")
```

Interface

Een interface is een soort class, maar dan zonder implementatie van code en met alleen de publieke methods, events en properties. Dit wordt vaak gebruikt indien een applicatie is opgedeeld in meerdere projecten (bijv. een EXE en een DLL) en waarbij elk project door verschillende ontwikkelaars wordt gebouwd. De ontwikkelaar van de DLL dient interfaces te definiëren voor de public classes die hij gaat maken. De ontwikkelaar van de EXE kan de EXE bouwen en al referenties leggen naar de DLL met behulp van de interfaces. De ontwikkelaar van de DLL kan hierna de interface implementeren en de classes voorzien van de functionaliteit.

```
Public Interface IPersoon
  Function ValideerGegevens() As String
End Interface
```

In dit geval kan de class Persoon de interface IPersoon implementeren via:

```
Public Class Persoon
  Implements IPersoon
End Class
```

User Defined Types

User Defined Types zijn zogenaamde ‘container types’. Een User Defined Type kan meerdere verschillende types van verschillende lengtes bevatten. Deze User Defined Types zijn in VB.NET vervangen door ‘Structures’. Structures bevatten echter veel meer functionaliteit dan User Defined Types. Een User Defined Type kan ook vervangen worden door een Class. De verschillen tussen structures en classes zit vooral in het geheugen gebruik en het feit dat classes inheritance en polymorphism ondersteunt. Het gaat te ver om de verschillen in deze Reference Guide verder uit te diepen. Mocht een User Defined Type uit een VB6 project vervangen moeten worden, dan is het aan te raden de verschillen verder te bestuderen. Zie literatuurlijst voor een verwijzing naar een whitepaper op het Internet.

Controls

Wat verder een groot verschil is tussen VB6 en VB.NET is dat alle controls ook als object gezien worden. Dit heeft als gevolg dat de default properties van alle controls verdwenen zijn. Immers indien in code `txtNaam` gebruikt wordt, interpreteert VB.NET dit als de control zelf en pas met de code `txtNaam.Text` interpreteert VB.NET de code als de ‘Text’ property van de control.

Bij het maken van een class zal gelijk opvallen dat de syntax voor het maken van een property is veranderd. In VB6 moest nog een aparte ‘Get’ en ‘Let’ of ‘Set’ gemaakt worden. In VB.NET is dit een statement geworden. In dit statement kan ook meegegeven worden of de property default en of read-only is.

```
Private _propertyValue As Integer = 0

Public Property MyProperty() As Integer

  Get
    Return _propertyValue
  End Get

  Set(ByVal value As Integer)
    ' Alleen waarden kleiner dan 10 zijn toegestaan.
    If value < 10 Then
      _propertyValue = value
    Else
      Throw New ApplicationException _
                ("Waarde mag niet kleiner zijn dan 10 !")
    End If
  End Set

  End Property
```

Solution

De projectgroep in VB6 heeft een andere naam gekregen in VS.NET. Een project bevindt zich nu altijd in een solution. Net zoals een projectgroep meerdere projecten kon bevatten, kan een solution ook meerdere projecten bevatten. Extra hierin is dat een solution niet persé alleen VB.NET projecten hoeft te bevatten. Zo kan in één solution een VB.NET Windows application, ASP.NET C# Web Service en een C++ Datacomponent bevatten.

Tabel. 4: VB6 projecttypes en de equivalenten in VB.NET

VB6 projecttype	VB. NET projecttype
Standard EXE	Windows application
ActiveX EXE	Geen equivalent (Kies tussen een Windows application of een Class Library)
ActiveX DLL	Class Library
ActiveX control	Windows control library
WebClass-based project	Web Forms

Projecteigenschappen

In de projecteigenschappen zijn een aantal onderdelen veranderd. De projectnaam in VB6 is vervangen door de ‘Assembly name’ en de ‘Root Namespace name’. De Assembly name wordt gebruikt voor het laden van de EXE of DLL van het project. De Root Namespace name is de naam waarnaar een referentie gemaakt kan worden door andere applicaties. De Root Namespace name is te vergelijken met de project naam in VB6.

3.2.2 Belangrijke toevoegingen VB.NET

Namespaces
Het .NET Framework is hiërarchisch opgebouwd, door middel van namespaces. Namespaces worden gebruikt om de grote hoeveelheid aan classes in het Framework te groeperen. Met een namespace kan een naam gegeven worden aan een groep classes.
Namespaces kunnen ook zelf gemaakt worden. Ze zijn ook handig om classes met dezelfde naam te kunnen gebruiken. Zo kan een class auto ook een subclass wiel hebben. Maar een class fiets kan ook een subclass wiel hebben. Toch zijn dit twee verschillende classes. Door middel van een namespace kan onderscheid gemaakt worden tussen beide classes.

Class object vervoermiddel.vb

```
Namespace Automobiel

  Public Class Automobiel
    'Hier code voor het auto object
  End Class

  Public Class Wiel
    'Hier code voor het wiel object van de auto
  End Class

End Namespace

Namespace Fiets

  Public Class Fiets
    'Hier code voor het fiets object
  End Class

  Public Class Wiel
    'Hier code voor het wiel object van de fiets
  End Class

End Namespace
```

Garbage Collector
Wat een grote toevoeging is van VB.NET ten opzichte van VB6 is de Garbage Collector. Deze zorgt voor het beheer van het geheugen en ruimt de objecten netjes op. Hierdoor is het gebruik van `Set obj = Nothing` niet meer nodig. Dit heeft performance verbetering tot gevolg en voorkomt memory leaks. Het kan natuurlijk nooit kwaad om de objecten netjes op te ruimen. Zeker tussendoor, wanneer het object niet meer nodig is, is het verstandig om het geheugen weer vrij te geven.
Let wel op, de Garbage Collector ruimt alleen de objecten uit .NET netjes op. COM objecten (VB6 DLL's) moeten dus nog steeds zelf opgeruimd worden.
Elk COM object erft de interface iDispose, zodat een COM object opgeruimd kan worden met `MyObject.Dispose`. Bovendien kan het laten uitvoeren van het opruimen van objecten door de Garbage Collector voor rare verrassingen zorgen. Zo is het niet meer te controleren wanneer de 'Finalize' method afgaat waarin bijvoorbeeld nog bestanden worden afgesloten etc. De 'Finalize' method gaat namelijk pas af bij het opruimen van het object.

Tasklist
Verder is een handige toevoeging in de IDE van VS.NET de 'Tasklist'. Code die qua syntax niet correct is, wordt niet meer afgestraft met een melding, maar wordt rood onderstreept (zoals de spellingscontrole van Word) en de foutieve regel wordt opgenomen in de Tasklist. Door op een regel in de Tasklist te dubbelklikken kan teruggesprongen worden naar code die nog afgemaakt moet worden. Verder kan een ontwikkelaar, door een regel commentaar toe te voegen dat begint met 'TODO, ook zelf taken toevoegen aan de Tasklist.

Exception handling
Met VB.NET verloopt de foutafhandeling anders dan in VB6. De 'On Error Goto' statements zijn vervangen door 'structured exception handling'. Voordeel van 'structured exception handling' is dat de foutafhandeling te 'nesten' is binnen een procedure en dat de code daardoor leesbaarder is.

Overigens ondersteunt VB.NET de 'On Error Goto' syntax nog wel voor backward compatibility, maar het is aan te bevelen de nieuwe 'structured exception handling' te gebruiken.
De basis syntax van de 'structured error handling' is als volgt:

```
Try
  'Code dat een fout kan veroorzaken.
Catch
  'Code om de fout af te handelen.
Finally
  'Code dat altijd uitgevoerd moet worden.
End Try
```

De 'Try' en' End Try' statements zijn verplicht. De Catch en Finally statements zijn niet verplicht, maar tenminste één van beide moet wel opgenomen worden. Het is mogelijk om meerdere Catch statements op te nemen zodat elke Catch blok een specifieke fout afhandelt.

In het Try blok komt de code te staan die de functionaliteit van de applicatie bevat. Dit is te vergelijken met de VB6 code onder 'On Error Goto ErrorHandler'. Op het moment dat er een fout ontstaat wordt de code in het Try blok beëindigd en gaat de code verder in het Catch blok. Dit is te vergelijken met de VB6 code onder de label 'ErrorHandler:'. Wat hiervan afwijkt is dat de fout nog afgevangen kan worden in het Catch blok. Moest dit in VB6 nog gebeuren met een Select Case of een If statement om het foutnummer heen, in VB.NET is het mogelijk om meerdere Catch statements op te geven. Omdat de code van boven naar beneden wordt doorlopen zal het eerste Catch blok worden uitgevoerd waaraan de fout voldoet. Dus de meest specifieke fouten moeten daarom bovenaan komen te staan. Tot slot kan een Finally blok toegevoegd worden. Dit block kan gebruikt worden om een melding aan de gebruiker te tonen, een log-file te vullen of om classes op te ruimen. In VB6 werd hiervoor vaak de code 'Resume Exit_' gebruikt waarna onder de label 'Exit_:' de laatste code werd uitgevoerd alvorens de functie te verlaten.

```
Dim a As Integer = 2147483647
Dim b As Integer = 0
Dim c As Integer = 0

Try
  a += 1
  a = b / c
Catch exp As DivideByZeroException
  Console.WriteLine("Error: Divide by zero")
Catch exp As OverflowException
  Console.WriteLine("Error: Overflow")
Catch exp As Exception
  Console.WriteLine("Error: " & exp.Message)
Finally
  Console.ReadLine()
End Try
```

In bovenstaande code dient het laatste Catch blok om andere errors dan de gespecificeerde errors in voorgaande catch blokken, af te handelen.

Met behulp van het Throw statement kan een error in een procedure worden doorgegeven naar een ‘hogere’ procedure, dus de procedure van waaruit de procedure waarin de fout optreedt werd aangeroepen. Het Throw statement is te vergelijken met de VB6 code ‘Err.Raise’.

```
Public Class Persoon

  Private _leeftijd As Integer

  Public Property Leeftijd() As Integer
    Get
      Leeftijd = _leeftijd
    End Get
    Set(ByVal value As Integer)
      If value >= 0 Then
        _leeftijd = value
      Else
      Throw New ApplicationException _
                ("Leeftijd mag niet negatief zijn.")
      End If
    End Set
  End Property

End Class
```

In bovenstaande Leeftijd property wordt een fout gegenereerd indien een negatieve waarde aan de property wordt toegekend, omdat een persoon geen negatieve leeftijd kan hebben. In onderstaande code krijgt de property Leeftijd de waarde –1. Dit levert dus een fout op in de property code. Vervolgens wordt daarom het Catch blok doorlopen in de onderstaande code.

```
Sub Main()
  Dim p as New Persoon()

  Try
    p.Leeftijd = -1
  Catch exp As Exception
    Console.WriteLine(exp.Message)
  End Try

End Sub
```

3.2.3 Valkuilen VB.NET

Eén van de veranderingen die mogelijk verwarring geeft is het feit dat VB.NET ook gecompileerde code gebruikt in de IDE omgeving. Dit in tegenstelling tot VB6. Hierom worden alle files gelijk bewaard op het moment dat ze aangemaakt worden in de IDE. Dit betekent echter ook dat in een meerlaags applicatie alle lagen regelmatig gecompileerd moeten worden. Gebeurt dit niet, dan kan het zijn dat een laag niet goed communiceert met een andere laag of zelfs compileerfouten geeft, terwijl de geschreven code wel correct is. Een project kan eenvoudig gecompileerd worden door de menuoptie : 'Build Solution' te gebruiken. Hiervoor moeten eerst de relaties (dependencies) tussen de verschillende projecten worden vastgelegd. Dit zorgt er dan automatisch voor dat wanneer een project gebuild wordt eerst de projecten gebuild worden waarvan het project afhankelijk is.

Het Framework biedt heel veel functionaliteit die voorheen in VB6 en zeker in ASP zelf geprogrammeerd moest worden. Het Framework is echter zo uitgebreid dat een ontwikkelaar goed moet kunnen zoeken om de door hem gewenste functionaliteit te vinden. Er bestaat daardoor dan ook het gevaar dat er te weinig gebruik gemaakt wordt van de functionaliteit die het Framework biedt. Het is daarom ook belangrijk dat een ontwikkelaar niet alleen opleiding krijgt in de ontwikkeltalen van .NET, maar ook in het gebruik van en over de functionaliteit van het .NET Framework.

Veel VB6 ontwikkelaars zijn onbekend met OOP. Dit betekent dat deze ontwikkelaars moeite zullen hebben om goede code te schrijven. Dat wil niet zeggen dat de code niet werkt, maar wel dat deze mogelijk slechter performt of slechter onderhoudbaar is.

3.2.4 Conversie VB6 naar VB.NET

VS.NET bevat een 'Upgrade Wizard' waarmee VB6 applicaties geconverteerd kunnen worden naar VB.NET. De werking van de Upgrade Wizard is zeer eenvoudig. Door een VB6 project te openen in VS.NET wordt de wizard automatisch gestart. Tijdens het doorlopen van de wizard wordt gevraagd waar het nieuw te creëren VB.NET project bewaard moet worden. Vervolgens wordt voorgesteld wat het projecttype is (ActiveX DLL, Standard EXE etc.). Er volgen nog een aantal opties, hiervoor kan allemaal de standaardwaarde geaccepteerd worden.

Nadat de knop 'Finish' is ingedrukt start het daadwerkelijke converteren. Afhankelijk van de grootte van het project kan dit een langere tijd in beslag nemen. De gedachte achter de Upgrade Wizard is dat de code zo min mogelijk aangetast moet worden. De code moet herkenbaar blijven voor de ontwikkelaar. Verder wordt alleen de code geconverteerd waarvan een goed equivalent aanwezig is in .NET. Bij code die nagenoeg geconverteerd kan worden, wordt een commentaarregel geplaatst en een suggestie gegeven met de meest geschikte code.

Ook bij code die niet geconverteerd kan worden, wordt een commentaarregel geplaatst. De ontwikkelaar is echter zelf verantwoordelijk voor de uiteindelijke conversie. Alle commentaarregels worden net zoals de eerder genoemde TODO items opgenomen in de Tasklist. De reden hiervoor is dat de ontwikkelaar de uiteindelijke beslissingen neemt over de conversie en zo controle blijft houden over zijn eigen applicatie.

De Upgrade Wizard kan de volgende items converteren:
- VB6 project naar equivalente Visual Basic.NET project;
- VB6 Forms naar equivalente Windows Forms;
- VB6 controls naar equivalente .NET Framework controls;
- ActiveX controls naar equivalente .NET Framework ActiveX controls;
- VB6 language statements naar VB.NET language statements (indien equivalent aanwezig);
- VB6 language statements naar VB.NET compatibility library class statements (indien equivalent niet aanwezig).

De Upgrade Wizard is niet geschikt voor het converteren van:
- ActiveX EXE projects;
- Add-In Designer, DHTML, Page Designer. DataReport, DataEnvironment;
- ActiveX controls: SSTab en UpDown;
- ActiveX document projects;
- OLE container control;
- Drag and Drop;
- Graphics statements en graphical controls;
- Dynamic Data Exchange (DDE).

Nadat een VB6 project is geconverteerd wordt er een Upgrade Report gegenereerd (zie ***Fig. 5***). In dit rapport wordt onderscheidt gemaakt tussen ‘Errors’ en ‘Warnings’. Zoals hiervoor reeds verklaard, probeert de Upgrade Wizard niet koste wat het kost het project te converteren. Onder ‘Errors’ kunnen de items gevonden worden die niet geconverteerd konden worden. Onder ‘Warnings’ staan de items die wel geconverteerd zijn, maar waarbij de .NET equivalent niet exact dezelfde functionaliteit bevat.

Upgrade Report for MyButton.vbp

Time of Upgrade: 9/5/2001 10:21 PM

List of Project Files

New Filename	Original Filename	File Type	Status	Errors	Warnings	Total Issues
⊞ (Global Issues)				0	0	0
⊟ MyButton.vb	MyButton.frm	Form	Upgraded with issues	6	1	7

Upgrade Issues for MyButton.frm:

#	Severity	Location	Object Type	Object Name	Property	Description
1	Compile Error	Form_Load	VB.CommandButton	cmdMyButton	UseMaskColor	VB.CommandButton property cmdMyButton.UseMaskColor was not upgraded.
2	Design Error	(Layout)				Data object was not upgraded.
3	Design Error	(Layout)	Data	Data1		Data control Data1 was not upgraded.
4	Design Error	(Layout)	VB.CommandButton	cmdMyButton	UseMaskColor	VB.CommandButton property cmdMyButton.UseMaskColor was not upgraded.
5	Design Error	(Layout)	VB.Form	Form1	ScaleHeight	VB.Form property Form1.ScaleHeight was not upgraded.
6	Design Error	(Layout)	VB.Form	Form1	ScaleWidth	VB.Form property Form1.ScaleWidth was not upgraded.
7	Runtime Warning	Data1_Validate	VB.Data	Data1	Validate	VB.Data Event Data1.Validate was not upgraded.

Fig. 5: Upgrade Report van de upgrade Wizard

Een conversie van een VB6 applicatie naar een VB.NET applicatie zal uiteindelijk leiden tot een werkende applicatie. Deze applicatie zal echter ook bestaan uit delen die geconverteerd zijn naar de VB compatibiliteitsclasse van .NET. Hierdoor wordt niet gebruik gemaakt van de functionaliteit van het Framework, waardoor uiteindelijk onderhoud veel meer tijd vergt. Bij grote applicaties is het gebruik van de wizard daarom niet aan te raden.

3.2.5 Visual Sourcesafe

VS.NET levert een nieuwe versie van Visual SourceSafe (VSS) mee. Ook deze is weer compleet geïntegreerd in VB.NET zoals ook bij VB6. In VB6 was dit versie 6.0b, dit is voor VB.NET versie 6.0c geworden. Dit suggereert een kleine update. Dit is echter niet het geval! Applicaties in een 6.0c database zijn niet meer te benaderen door VB6. Bovendien kan versie 6.0c niet geïnstalleerd worden naast versie 6.0b. Gelukkig wordt dit ook gemeld voordat de installatie gestart wordt. Klik deze melding niet zomaar weg. Voor ontwikkelaars die zowel VB6 applicaties als .NET applicaties bouwen is het verstandiger om gebruik te blijven maken van VSS 6.0b. Deze wordt namelijk ook ondersteund door .NET.

Een verandering in het gebruik van VSS in .NET is dat er bij het bewaren van een project niet meer gevraagd wordt of het project toegevoegd moet worden aan VSS. Dit komt, doordat alle project files bij het aanmaken al op de harde schijf bewaard worden. Ze worden bij het bewaren echter niet automatisch in Sourcesafe geplaatst. Het toevoegen van een project of een complete solution is dan ook een handmatige actie geworden. Deze kan uitgevoerd worden met de menuoptie ‘File – Source Control – Add Solution to Source Control’. Op het moment dat een solution in VSS is gezet, zullen alle projecten die aan deze solution worden toegevoegd automatisch ook in VSS gezet worden. Hierbij hanteert VSS dezelfde projecten structuur zoals deze in de solution explorer (en ook in Windows verkenner) te zien is.

3.3 ASP.NET

ASP.NET is een ontwikkeltaal voor het bouwen van Web applicaties en Web Services en is de opvolger van ASP. Voordat ASP.NET en het verschil tussen ASP en ASP.NET beschreven kan worden is het eerst van belang te weten hoe ASP is ontstaan en wat het precies is.
Bij het ontstaan van internet pagina’s werd HTML de standaard lay-out taal. Het nadeel van HTML is echter dat het statische pagina’s oplevert. Met de toevoeging van Javascript was interactie met de gebruiker al meer mogelijk. Ook met de opkomst van DHTML was het mogelijk om interactieve pagina’s te maken. DHTML ondersteunt ook ActiveX, nadeel hiervan is dat dit altijd op de client uitgevoerd moet worden en dit alleen ondersteund wordt in Internet Explorer van Microsoft. Echt dynamische pagina’s maken werd mogelijk met de introductie van ASP. ASP wordt namelijk server-side uitgevoerd. Daarnaast zijn er allerlei andere webtalen zoals PHP die ook server-side pagina’s ondersteunen. Met behulp van ASP is het mogelijk om connecties te maken naar databases, XML documenten etc. Deze data kan op de server worden geïnterpreteerd en dynamisch wordt de pagina met deze gegevens gevuld. De client krijgt uiteindelijk de gegenereerde HTML pagina te zien, zonder dat er nog te achterhalen is, welke code gebruikt is. Dit in tegenstelling tot onder andere Javascript waar de code altijd meegestuurd wordt naar de client. De meest gebruikte scripttaal achter ASP is VBscript. Een groot nadeel van ASP is dat de programmeertaal VBscript en de lay-outtaal HTML door elkaar gebruikt worden. Dit bemoeilijkt de onderhoudbaarheid van Web applicaties.

In ASP.NET zijn de goede dingen van ASP gebleven en de gebreken en de beperkingen van ASP aangepakt. In de komende paragrafen wordt dit verder verklaard.

3.3.1 Verschil ASP en ASP.NET

Allereerst de overeenkomsten tussen ASP en ASP.NET. Ook bij ASP.NET wordt de code op de server uitgevoerd en is het nog steeds mogelijk om ASP en HTML door elkaar heen te gebruiken.

Een groot verschil is het aantal controls dat ASP.NET ter beschikking stelt. Was het nog zo dat in ASP gebruik gemaakt moest worden van de standaard HTML <FORM> controls, ASP.NET beschikt over haar eigen Web Forms class library, met een groot aantal controls. Deze kunnen nog een keer uitgebreid worden middels de Microsoft Internet Explorer WebControls. Deze uitbreiding bevat onder andere een treeview control, een toolbar control en een tabstrip / multipage control. De uitbreiding is te downloaden op de Microsoft pagina :

http://msdn.microsoft.com/downloads/samples/internet/asp_dot_net_servercontrols/webcontrols/default.asp

3.3.2 Belangrijke toevoegingen ASP.NET

De hiervoor genoemde Web Forms Class Library moet ook gezien worden als een toevoeging. Met behulp van deze Class Library is het mogelijk om web pagina's te maken die qua functionaliteit niet onder doen voor de traditionele desktop applicaties.

Code Behind files

Een hele belangrijke toevoeging in ASP.NET is de toevoeging van zogenaamde 'Code-Behind files'. Deze Code-Behind files kunnen vergeleken worden met de module zoals deze achter een Windows Form zit. De code en de lay-out zijn nu van elkaar gescheiden. Hierdoor onstaat duidelijkere code en kunnen events van controls duidelijk verwerkt worden.
Verder is een belangrijke verbetering dat ASP.NET in plaats van een scripttaal een volwaardige ontwikkeltaal ondersteunt. Bij het maken van een ASP.NET applicatie kan gekozen worden uit de standaard aanwezige ontwikkeltalen VB.NET of C#.

.NET Framework

Het .NET Framework biedt een groot aantal classes waarvoor voorheen nog 3rd-party componenten nodig waren. Deze classes bieden o.a. functionaliteit voor XML, data access, file upload, regular expressions, image generation, performance monitoring en logging, transacties, message queuing, SMTP mail. Doordat ASP.NET onderdeel uitmaakt van het Framework kan ook direct van de andere classes gebruik gemaakt worden.

Performance en schaalbaarheidsvoordelen

ASP.NET is veel sneller dan ASP. ASP.NET detecteert automatisch veranderingen, compileert zelf bestanden indien nodig en slaat gecompileerde resultaten op voor hergebruik. Dankzij de compilatie van code is de performance een stuk hoger.
ASP.NET output caching kan de performance en schaalbaarheid van een webapplicatie verbeteren. Indien output caching aan staat voor een pagina, voert ASP.NET de pagina één keer uit en bewaart het resultaat in geheugen. Wanneer een andere gebruiker dezelfde pagina opvraagt stuurt ASP.NET het gecachedte resultaat vanuit het geheugen naar de gebruiker zonder de code dus weer uit te voeren. Output caching kan worden ingesteld voor de gehele pagina of voor een deel van de pagina.

Betrouwbaarheid
ASP.NET bevat zogenaamde Memory Leak, DeadLock en Crash Protection. ASP.NET detecteert en herstelt automatisch problemen met deadlocks en geheugenverlies zodat de webapplicatie altijd beschikbaar is voor gebruikers.

Installatie van webapplicaties
Een ASP.NET applicatie wordt geïnstalleerd door het eenvoudig te kopiëren naar de web server. Met ASP.NET kunnen gecompileerde componenten worden geüpdate zonder dat de web server herstart moet worden. Het kopiëren van de component over de bestaande DLL volstaat. ASP.NET ziet automatisch dat er veranderingen zijn opgetreden en zal de nieuwe code gebruiken.
Bestaande ASP applicaties kunnen blijven draaien naast ASP.NET applicaties. Zelfs binnen één applicatie kunnen zowel traditionele .asp bestanden als .aspx (.NET) bestanden bestaan.

Custom controls
Binnen ASP.NET is het nu ook mogelijk om zelf controls te bouwen. Hierbij kan onderscheid gemaakt worden tussen user controls en custom controls.
Een user control (ook wel een pagelet genoemd) is een control die vergeleken kan worden met een ActiveX OCX. Een user control heeft de extensie .ASCX. Op een user control kunnen allerlei controls gesleept worden, zodat een soort sub pagina gemaakt wordt. Een voordeel hiervan is dat bijvoorbeeld een pagina header met een bepaald logo op deze manier snel op elke pagina geplakt kan worden. Een nadeel van user controls is dat ze alleen binnen een project gebruikt kunnen worden en dat ze een slechte design time lay-out hebben. Voordeel van een user control is dat hij snel te maken is.
Custom controls daarentegen hebben een mooie design time lay-out, maar zijn weer lastiger te programmeren. Custom controls hebben namelijk geen ‘user interface’ waarop controls gesleept kunnen worden. Alle controls moeten met behulp van code geïnstantieerd worden. Ook de methode om HTML te genereren moet zelf geprogrammeerd worden. Een groot voordeel is dat Custom controls over meerdere projecten heen gebruikt kunnen worden. Ze zijn dan ook geschikt om op te nemen in een library.

Trace
ASP.NET pagina’s kunnen met behulp van de trace optie geanalyseerd worden. Hierbij kan onder andere de performance van de pagina bekeken worden. Boven aan de pagina wordt een trace gedeelte toegevoegd met hierin onder andere de volgende informatie: SessionID, header van de pagina met daarin de aanwezige cookies, de volgorde van het opbouwen van de controls, laadtijd en de server variabelen. De trace optie wordt gestart door deze in de Page tag van de pagina aan te zetten.

```
<%@ Page trace=”True” %>
```

Het is ook mogelijk om functionaliteit, zoals `Debug.Print` of `Debug.Assert` (bekend van VB6) op te nemen. Dit kan met behulp van :

```
Trace.Write(“Button clicked”)
Trace.Warn(“Value: “ + value)
```

Het is ook mogelijk om pagina’s te tracen zonder dat de gebruiker de output hiervan in zijn scherm krijgt. Dit kan door in de web.config file de trace optie aan te zetten.

```
<configuration>
  <system.web>
    <trace enabled="true" requestlimit="10"/>
  </system.web>
</configuration>
```

Op de webserver wordt vervolgens een pagina in het geheugen opgebouwd. Deze is dus niet terug te vinden op de harde schijf van de webserver. De pagina kan via de volgende URL geopend worden http://servername/approot/Trace.axd. Het betreft hier dus een verwijzing naar het geheugen.

3.3.3 Valkuilen ASP.NET

Een belangrijke valkuil is de ontwikkeltijd van web applicaties. Ten opzichte van het maken van web applicaties in ASP met behulp van bijvoorbeeld Visual Interdev neemt de ontwikkeltijd wel af en is ASP.NET een grote vooruitgang. Ten opzichte van Desktop applicaties echter kost een ASP.NET applicatie meer tijd. Dit komt vooral door de uitgebreidere en ingewikkeldere infrastructuur en door het gemis van sommige controls. Een goed voorbeeld hiervan is de menu-control van VB. Deze zit niet in ASP.NET. Om een menu te maken zal dus zelf geprogrammeerd moeten worden. Het verschil in ontwikkeltijd voor Web applicaties kan hierdoor al gauw oplopen tot een factor twee verschil met een Desktop applicatie.

Controls

Naast de menu-control wordt ten opzichte van VB.NET voornamelijk de controls treeview, tabstrip en toolbar gemist. Deze controls worden niet standaard door Microsoft meegeleverd, terwijl wij ze wel regelmatig gebruiken worden gebruikt binnen het MNP. Deze controls worden echter door Microsoft (unsupported) en third party leveranciers (meestal tegen betaling) aangeboden. Omdat er binnen het MNP heel vaak gebruik gemaakt wordt van de treeview is deze control extra getest. Er is ook een treeview gevonden waarbij het mogelijk is om checkboxen te gebruiken. Tijdens de pilot bleek deze echter nog de nodige bugs te bevatten.
De volgende drie treeview controls zijn getest

- Microsoft.Web.UI.webcontrols.treeview
  Dit is een gratis Microsoft-control die hetzelfde werkt als de VB.NET component. Daarom is het meest voor de handliggende om deze control te gebruiken.
  Er zit echter een grote bug in deze component waardoor hij niet geschikt is gevonden hem te gebruiken. Deze control werkt namelijk niet in IE6, maar slechts alleen goed in IE5.
- Obout_ASPtreeview.treeclass
  Met deze control kan een treeview gedisplayed worden en na een keuze doorlinken naar een andere pagina. Met deze control is het niet mogelijk om het geselecteerde item op te vragen. Het is al helemaal niet mogelijk om checkboxen aan te zetten.
- VisualAspv4treeview
  Deze treeview heeft dezelfde beperking als de treeview van obout.

Bound / Unbound

De meeste controls uit de ASP.NET class library zijn unbound. Dit betekent dat een control niet rechtstreeks aan een Dataset kan worden gekoppeld. Het heeft dus niets met statefull te maken, aangezien een dataset nooit een directe koppeling heeft met een database. Een grote beperking dat een dataset niet aan een control gekoppeld kan worden is dat bij het bladeren door een dataset niet automatisch de waarde van de control bijwerkt. Er moet extra code geschreven worden om de control met de goede waarde te vullen. Het is wel mogelijk om een

listbox, combobox en datagrid te koppelen aan een dataset. Datasets zijn alleen aanwezig in het geheugen. Meer informatie over datasets is te vinden in paragraaf 3.5.1.
Een oplossing om controls toch aan een waarde te koppelen is door in de ASP tag van de control hier code voor op te nemen. Doordat dit niet in een property veld kan worden geregeld blijft het toch behelpen.

3.3.4 ASP.NET Web Matrix

Specifiek voor het bouwen van ASP.NET applicaties heeft Microsoft een gratis ontwikkelomgeving gebouwd. Dit programma, Web Matrix genaamd is te downloaden vanaf het Internet op de ASP.NET homepage. http://www.asp.net/. Een groot nadeel van Web Matrix ten opzichte van de IDE van VS.NET is onder andere dat Web Matrix geen gebruik maakt van IntelliSense. Deze en andere eigenschappen van IDE maken dat Web Matrix geen alternatief is voor degenen die beschikken over VS.NET. Voor degenen die VS.NET niet hebben is het een goedkoop bruikbaar alternatief.

3.4 XML

3.4.1 Wat is XML?

XML staat voor eXtended Markup Language. XML was in eerste instantie bedacht om data te beschrijven. XML bevat dus niet alleen data, maar ook informatie over deze data. XML is tot standaard verheven en is gedeponeerd bij het W3C consortium. Het W3C consortium is een organisatie waarin de meeste grote bedrijven zoals Microsoft, Oracle, Sun, etc. in vertegenwoordigd zijn en die zorgt voor het standaardiseren van het Internet en alles hieromheen. Dit heeft als groot voordeel dat alle softwarebedrijven hun software geschikt maken voor één taal XML en er dus geen Microsoft, Oracle, SUN etc. dialecten ontstaan. Het W3C heeft XML verheven tot de standaard uitwisseltaal voor data.

Syntax

XML is een MarkUp taal en bestaat dus uit data en metadata. De data is de tekst die de waarde bevat, de metadata zijn de tags die de data beschrijven. De metadata geeft dus extra informatie over de waarde. Dit is goed te vergelijken met het onderwaterscherm, van WP 5.1. Echter de syntax van XML is veel strenger.
Een voorbeeld van een XML document is:

```
<?xml version="1.0"?>
<zin>
  <team>IMP</team> is een team van de
  sector <sector>MNP</sector> binnen het
  <bedrijf>RIVM</bedrijf>.
</zin>
```

In bovenstaand voorbeeld is nu bekend dat de waarde IMP voor een team staat, de waarde MNP voor een sector en de waarde RIVM voor een bedrijf. Door het koppelen van een Cascading StyleSheet aan een XML file kan de opmaak van een tag bepaald worden, zodat bijvoorbeeld de tag ‘team’ altijd cursief en de tag ‘sector’ altijd onderstreept wordt afgedrukt.

IMP is een team van de sector MNP binnen het RIVM.

Een van de voorwaarden van de syntax van XML is dat alle waarden binnen een tag moeten staan. Vandaar dat in bovenstaand voorbeeld de tags ‘zin’ zijn toegevoegd. Belangrijk bij tags is dat ze ‘case sensitive’ zijn. Ook mogen tags niet door elkaar gebruikt worden. Elke tag

moet in de omgekeerde volgorde afgesloten worden als dat ze zijn begonnen. Het logo van RIVM mag dus niet omschreven worden als:

```
<cursief><wit>ri</cursief>vm</wit>
```

Ondanks dat het klopt wat er staat (de letters ‘ri’ zijn cursief en wit en de letters ‘vm’ zijn alleen wit) zal dit toch een fout opleveren. De juiste syntax is:

```
<wit><cursief>ri</cursief>vm</wit>
```

In de hiervoor genoemde voorbeelden wordt gebruik gemaakt van zelf verzonnen tags. Omdat de syntax van XML zeer strikt is, moeten alle tags beschreven zijn. Dit kan met behulp van een zogenaamde DTD of een XSD. Dit is verder beschreven in paragraaf 3.4.2 respectievelijk paragraaf 3.4.3.

Indien er binnen een tag meerdere tags gebruikt worden, wordt ook wel gesproken van elementen. Elk element kan ook weer een aantal sub elementen bevatten.

```
<?xml version="1.0"?>
<team>
  <afkorting>IMP</afkorting>
  <naam>Informatievoorziening en Methodologie
        Planbureau</naam>
  <gebouw>W</gebouw>
</team>
```

Tags kunnen ook bestaan uit attributes. Dit zijn onderdelen van een tag die de data in de tag zelf beschrijven. De waarde van een attribute moet altijd tussen dubbele quotes staan.

```
<?xml version="1.0"?>
<team afkorting=”MNP”>
  <naam>Informatievoorziening en Methodologie
        Planbureau</naam>
  <gebouw>W</gebouw>
</team>
```

Ondanks dat bovenstaand voorbeeld dezelfde gegevens bevat als het eerder gegeven voorbeeld is het toch aan te raden zoveel mogelijk gebruik te maken van elementen i.p.v. attributen. Een aantal redenen hiervoor zijn :

- attributen kunnen niet meerdere waarden bevatten;
- attributen zijn lastig vanuit code te bewerken;
- attribuutwaarden kunnen lastig tegen een DTD gevalideerd worden. Het begrip DTD wordt verder uitgelegd in paragraaf 3.4.2.

Een voorbeeld waarin een attribuut wel handig kan zijn is voor het opslaan van een ID. De waarde heeft dan niet veel betekenis voor degene die de data moet lezen, maar kan wel handig zijn voor een ontwikkelaar.

3.4.2 DTD

Met behulp van Document Type Definition (DTD) kan de definitie van eigen gemaakte tag vast gelegd worden. In een DTD worden alle mogelijke tags, de attributen van de tags en de mogelijke combinaties beschreven.

Er bestaan op het Internet reeds een aantal DTD's voor specifieke toepassingen. Dit zijn onder andere:

- DocBook:
  om gestructureerde documenten te maken;
- MathML:
  om wiskundige symbolen en formules mee te schrijven;
- RDF (Resource Description Framework):
  om metadata te coderen en te hergebruiken;
- SOAP (Simple Object Access Protocol):
  om processen met elkaar te laten communiceren (data uitwisselen en 'remote procedure calls' doen).
- SVG (Scalable Vector Graphics):
  om plaatjes, scripts en animatie mee te beschrijven (.svg bestanden worden algauw heel groot, ook veel gebruikt zijn .svgz bestanden. Die zijn met het gzip algoritme gecomprimeerd);
- XHTML:
  XHTML is de XML variant van HTML versie 4.01.

Het is mogelijk om verschillende DTD's door elkaar te gebruiken. In dat geval is het van belang om per tag aan te geven welke DTD gebruikt wordt. Dit gebeurt door de namespace van de DTD op te nemen in de tag. In het volgende voorbeeld wordt specifiek de namespace van de DTD's 'kleur' en 'opmaak' opgegeven.

```
<kleur:wit><opmaak:cursief>ri</opmaak:cursief>vm</kleur:wit>
```

Met behulp van dezelfde namespaces kan de tekst dus ook veranderd worden in:

```
<kleur:groen><opmaak:vet>ri</opmaak:vet>vm</kleur:groen>
```

Het gaat te ver om hier de gehele syntax van DTD te beschrijven. Wel is het belangrijk dat in een DTD ook beschreven is uit welke data een tag mag bestaan, of een element van een tag verplicht is, of dat hij ook leeg gelaten mag worden, en of er meerdere waardes in de elementen in mogen staan. Een DTD kan dus ook gebruikt worden om een XML document te valideren.
De syntax van een DTD bestand ziet er als volgt uit.

```
<!ELEMENT team (afkorting, naam, gebouw)>
<!ELEMENT afkorting (#PCDATA)>
<!ELEMENT naam (#PCDATA)>
<!ELEMENT gebouw (#PCDATA)>
```

Een XML document wordt op de volgende manier aan een DTD bestand gekoppeld.

```
<?xml version="1.0"?>
<!DOCTYPE team SYSTEM "team.dtd">

<team>
  <afkorting>MNP</afkorting>
  <naam>Milieu en Natuur Planbureau</naam>
  <gebouw>W</gebouw>
</team>
```

3.4.3 XSD

XSD staat voor Xml Schema Definition, vaak wordt ook wel de term XML Schema Language of kortweg XML Schema gebruikt. XSD kan wel gezien worden als de opvolger van DTD. Een XSD bestand definieert de volgende eigenschappen van een XML document:

- de elementen die kunnen voorkomen in een XML document;
- de attributen die kunnen voorkomen in een XML document;
- de relatie tussen elementen en child-elementen;
- de volgorde en aantal child-elementen;
- de eigenschappen van elementen zoals, mag het element leeg zijn, het data type, default waarde, vaste waarden.

De verschillen tussen DTD en XSD is:

- XML Schema's zijn geschikt gemaakt voor mogelijke uitbreidingen in de toekomst;
- XML Schema's zijn rijker en meer bruikbaar dan DTDs;
- XML Schema's zijn geschreven in XML;
- XML Schema's ondersteunen andere XML Schemas en namespaces;
- XML Schema's zijn als standaard geaccepteerd door het W3C consortium.

In de vorige paragraaf is een voorbeeld gegeven van een DTD bestand. Hieronder is een voorbeeld van dezelfde gegevens maar dan in de XML Schema Language.

```
<?xml version="1.0"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://dotnetPilot.rivm.nl/"
  xmlns="http://dotnetPilot.rivm.nl/elementFormDefault=
        "qualified">
    <xs:element name="team">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="afkorting" type="xs:string"/>
          <xs:element name="naam" type="xs:string"/>
          <xs:element name="gebouw" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Ook de referentie naar een XML schema verschilt van een referentie naar een DTD.

```
<?xml version="1.0"?>
<team
xmlns="http://dotnetPilot.rivm.nl/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://dotnetPilot.rivm.nl/schema/team.xsd">

<team>
  <afkorting>MNP</afkorting>
  <naam>Informatievoorziening en Methodologie
        Planbureau</naam>
  <gebouw>W</gebouw>
</team>
```

Er bestaan allerlei programma's waarmee XML schema's gegenereerd kunnen worden. Dit kan met Visual Studio.NET zelf, maar een veel gebruikt niet Microsoft programma voor alles

wat met XML te maken heeft is XML Spy (http://www.xmlspy.com). Dit programma is zeer geschikt om te gebruiken wanneer er met XML documenten gewerkt wordt.

3.4.4 DOM object

Het XML Document Object Model (DOM) is een interface om XML documenten gestructureerd vanuit code te benaderen. Het DOM is onderdeel van de XML Parser, welke geïnstalleerd kan worden op een PC. (Dit is te vergelijken met ADO dat onderdeel is van MDAC). Het DOM bestaat uit een aantal objecten zoals onder andere Nodes, Attributes en ParserError. De eerste is een synoniem voor de elementen in een XML document en de tweede voor de attributen van een element. Het object ParserError is een equivalent van het Err object van VB. Bij het laden en wegschrijven van XML bestanden kan met behulp van het ParserError object eventuele fouten worden getraceerd. Verder heeft het DOM veel methods, waaronder Load en LoadXML om een XML document in te lezen, c.q. een XML string in te lezen.

```
Dim docteam As New XMLDOM

docteam.async=”false”
docteam.load(“C:\team.xml”)
```

Ook beschikt het DOM over het HTTP request object. Hiermee is het mogelijk om op een client een request uit te voeren op een webserver. Op die manier kan bijvoorbeeld een xml document van een webserver geladen worden.

```
Dim reqTeam As New XMLHTTPrequest
Dim xmlDocument As String

reqTeam.open("GET", "team.xml", false)
reqTeam.send()

xmlDocument= reqTeam.responseText
```

Om meer inzicht te krijgen in de mogelijkheden van het DOM object kan het beste de Objectbrowser gebruikt worden. Hierin zijn alle properties en methods beschreven.

3.4.5 XSL / XSLT

eXtensible Stylesheet Language (XSL) is een taal voor het bouwen van lay-out voor het presenteren van XML documenten. Door een XML document te koppelen aan een XSL bestand kan een XHTML pagina gegenereerd worden. Met behulp van de XSL kan vastgelegd worden hoe de tags van XML door een browser geïnterpreteerd moeten worden. Zo kan `<table>` een tabel in HTML betekenen, maar het kan ook een tag van XML zijn voor een element ‘tabel’. XSL is ook weer als standaard gedeponeerd bij het W3C consortium. Niet elke browser ondersteunt echter volledig een XHTML bestand dat gegenereerd is met een XSL bestand. Zo werkt het nog niet goed in IE5 en IE5.5, maar wordt het pas goed ondersteund in IE6. Ook Netscape 6 ondersteunt XSL niet volledig. XSL bestaat uit drie delen, te weten: XSLT, XPath, en XSL Formatting Objects.

XSLT staat voor XSL Translation en bevat de daadwerkelijke vertaalslag van een XML document naar een XHTML bestand. Een XSL style sheet begint altijd met het root element `<xsl:stylesheet>` of `<xsl:transform>`. Beide elementen zijn gelijk aan elkaar en kunnen allebei gebruikt worden.

Bij gebruik van IE6 of Netscape 6 moet een XSL als volgt begonnen worden.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

of:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Overigens moet voor IE5 nog een oude koppeling naar een stylesheet gemaakt worden. Te weten:

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

Als voorbeeld van het genereren van een XHTML bestand wordt het eerder genoemde team XML document gebruikt, uitgebreid met een aantal extra teams. Hieronder wordt getoond hoe het XML document en het XSL bestand eruit zien en wat voor XHTML bestand daaruit voort komt.

XML bestand

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="team.xsl"?>

<bedrijf>
  <team>
    <afkorting>IMP</title>
    <naam>Informatievoorziening en Methodologie
          Planbureau</naam>
    <gebouw>W</gebouw>
  </team>
  <team>
    <afkorting>NMD</title>
    <naam>Nationale Milieubeleidsevaluatie en
          Duurzaamheid</naam>
    <gebouw>W</gebouw>
  </team>
.
.
.
</bedrijf>
```

XSL bestand

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>Team Informatie</h2>
    <table border="1">
    <tr bgcolor="#cccccc">
      <th align="left">Afkorting :</th>
      <th align="left">Naam :</th>
      <th align="left">Gebouw :</th>
    </tr>
    <xsl:for-each select="bedrijf/team">
    <tr>
      <td><xsl:value-of select="team"/></td>
      <td><xsl:value-of select="naam"/></td>
      <td><xsl:value-of select="gebouw"/></td>
    </tr>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

XHTML bestand

In de browser zal in de URL de naam van het XML document te zien zijn. Ook als er in de source van de browser gekeken wordt zal het XML document te zien zijn. Echter in de browser zelf zal ongeveer het volgende getoond worden.

Team Informatie

Team	Naam	Gebouw
IMP	Informatievoorziening en Methodologie Planbureau	W
NMD	Nationale Milieubeleidsevaluatie en Duurzaamheid	W

3.4.6 XML Databases

Het is ook mogelijk om XML documenten te gebruiken als database. Ook relaties tussen verschillende XML documenten en data in een XML document kunnen worden vastgelegd. Met de tool XQuery is het mogelijk om met een soort SQL dialect queries uit te voeren op zo'n XML database. Deze databases worden vaak aangeduid met de term Native XML Database.
Databases zoals SQL Server 2000, Oracle 9i en Access XP worden ook wel XML databases genoemd, aangezien ze geïntegreerde functies hebben om XML documenten te importeren, te exporteren, te interpreteren en op te slaan in XML formaat.

3.5 ADO.NET

In VB6 werd voor het benaderen van o.a. databases gebruik gemaakt van ActiveX Data Objects (ADO). In VB.NET gebeurt dit met ADO.NET, onderdeel van het .NET Framework. In ADO.NET wordt de nadruk gelegd op een gedistribueerd programmeermodel dat geschikter is voor de ontwikkeling van web applicaties. Dit heeft tot gevolg dat ADO.NET,

zoals reeds genoemd, alleen forward-only, read-only cursors ondersteunt voor het lezen van data. Stateful server-side cursors worden niet meer gebruikt.

3.5.1 Verschillen ADO en ADO.NET

Wat als eerste opvalt is het verdwijnen van het object Recordset. ADO bestaat uit vijf objecten, te weten Connection, Command, Parameter, Recordset en Error. In ADO.NET is het Recordset object vervangen door een drietal andere objecten, te weten: DataSet, DataReader en DataAdapter. In de volgende paragrafen worden deze objecten en hun verschillen met het Recordset object verder beschreven.

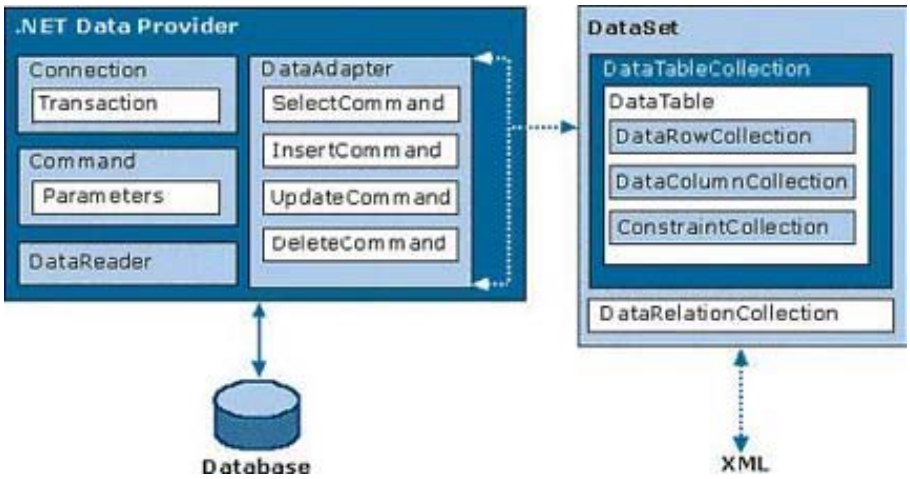


Fig. 6: Objectmodel ADO.NET

Dataset

- Structuur: Een ADO Recordset is te vergelijken met één tabel. Als een Recordset data bevat van meerdere database tabellen dan moeten deze gegevens met een JOIN query geselecteerd worden zodat één resultaattabel ontstaat die in de recordset wordt geplaatst Een DataSet daarentegen is een collectie van één of meer tabellen c.q. queries (views). De tabellen in een DataSet zijn DataTable objecten. Indien een DataSet gegevens bevat van meerdere database tabellen kan het dus meerdere DataTable objecten bevatten. Een DataSet kan tevens relaties tussen DataTable objecten bevatten. Dit betekent dat in een DataSet de structuur van de onderliggende database kan worden meegenomen.
- Data Navigatie: in ADO kan je sequentieel door de rijen van een recordset lopen met behulp van de MoveNext methode. In ADO.NET worden rijen gepresenteerd als een collectie. Dit betekent dat de rijen kunnen worden doorlopen zoals je een collectie doorloopt en een bepaalde rij kan geselecteerd worden via een index (ordinale of primary key index). DataRelation objects bevatten informatie over ‘master’ en detail records, zodat eenvoudig records behorende bij een bepaalde ‘master’ record geselecteerd kunnen worden.
- Cursor: een cursor is een database element voor recordnavigatie, update van data, en het zichtbaar maken van datawijzigingen in de database door andere gebruikers. ADO.NET bevat data classes die de functionaliteit van de traditionele cursor verstrekken, bijvoorbeeld de functionaliteit van een forward-only, read-only cursor is beschikbaar in het ADO.NET DataReader object.

DataReader

Een ADO.NET DataReader wordt gebruikt voor ‘read-only’, ‘forward-only’ selectie van data uit een database. Indien mogelijk kan door het toepassen van een DataReader i.p.v. een dataset de performance van de applicatie worden verbeterd en wordt het systeem minder zwaar belast omdat er altijd maar één rij in het geheugen wordt geladen. De DataReader wordt gecreëerd m.b.v. een Command object.

Het is mogelijk om meerdere selecties van data in één DataReader object in te lezen. Deze datasets kunnen in de volgorde waarin ze zijn aangemaakt worden doorlopen. In tegenstelling tot een Dataset zijn er geen relaties aan te geven tussen de verschillende datasets

DataAdapter
Nieuw in ADO.NET ten opzichte van ADO is de DataAdapter. Deze control kan gekoppeld worden aan één of meerdere Command object(en) en genereert een dataset. Het Command object heeft weer een koppeling met een Connection object, waardoor de dataadapter gekoppeld is aan een database. Een groot voordeel van de dataadapter is dat hij bijhoudt wat er veranderd is in de dataset. Gewijzigde, toegevoegde en verwijderde records worden gemarkeerd in de dataset. De dataadapter genereert zelf SQL statements voor het toevoegen, bijwerken en verwijderen van records. Met de method Update wordt alles in de database bijgewerkt, zonder dat de ontwikkelaar zich moet afvragen wat er is veranderd in de dataset en hiervoor één of meerdere sql statements moet aanroepen. Ook kunnen er tussen meerdere dataadapters relaties gelegd worden. Dit zorgt ervoor dat parent-child tabellen goed worden bijgewerkt. Daarnaast is het ook mogelijk om bijvoorbeeld alleen de gewijzigde records te updaten.

Connectie
Een ander groot verschil tussen ADO en ADO.NET is dat in ADO.NET een connectie na het uitvoeren van een Select of Update statement automatisch weer wordt gesloten. Een dataset is niet verbonden met de Data Source. Ook met ADO kan gewerkt worden met ‘disconnected’ recordsets maar ADO is vooral bedoeld voor recordsets waarbij de recordset steeds verbonden is met de data source.

Communicatie
In ADO wordt gecommuniceerd met de database via een OLE DB provider. In ADO.NET loopt de communicatie via een data adapter (een OleDbDataAdapter of een SqlDataAdapter object), die een OLE DB provider aanroept of de API’s van de onderliggende data source (SQL Server). Het belangrijkste verschil is dat de ontwikkelaar meer controle heeft op de wijze waarop wijzigingen in de dataset worden doorgegeven aan de database om hiermee de performance te optimaliseren, validatie uit te voeren of andere extra taken toe te voegen.

Data uitwisselen
Het uitwisselen van een ADO.NET dataset tussen applicaties is gemakkelijker dan het uitwisselen van een ADO disconnected recordset. Om een ADO disconnected recordset van de ene component naar door te geven wordt COM marshalling gebruikt. Bij ADO.NET wordt een dataset gebruikt die een XML stream kan uitwisselen. Dit heeft een aantal voordelen ten opzichte van COM marshalling.

- Geavanceerdere datatypen: COM marshalling kan een beperkt aantal datatypen aan. Omdat de uitwisseling van datasets in ADO.NET is gebaseerd op XML, is er geen restrictie voor datatypen.

- Performance: in tegenstelling tot ADO vereist ADO.NET geen datatype conversie. ADO vereist een datatype conversie naar de COM datatypen.
- Firewalls: een Firewall kan de uitwisseling van een disconnected ADO recordset tegengaan. Firewalls worden meestal zodanig geconfigureerd dat HTML text gewoon doorlaat en system-level verzoeken (zoals COM marshalling) niet. Omdat in .NET Datasets uitgewisseld worden (Dataset=XML) laten firewalls datasets wel passeren.

Gedrag
Het gedrag van een ADO Recordset object hangt af van de eigenschappen van de Connection, Command en Recordset object en van de functionaliteit van de database. Zo kan een Recordset wel of niet forward-only zijn, kan de data in een Recordset wel of niet te updaten zijn, en kan wel of niet een permanente connection met de database bestaan.
Met ADO.NET moet een bepaald object gebruikt worden afhankelijk van hoe de data gebruikt wordt. Als je maar één keer door de data heen loopt waarbij de data niet wordt veranderd, kan de ExecuteReader methode van het Connection object worden aangeroepen die een DataReader retourneert. Moet het mogelijk zijn door de data heen te ‘scrollen’ dan kan de Fill methode van het DataAdapter object worden gebruikt waarmee de data in een DataSet worden gezet. Wijzigingen in de data kunnen vervolgens met de Update methode van het DataAdapter object naar de database worden gestuurd. In ADO.NET is de functionaliteit van een object dus niet afhankelijk van bepaalde properties maar van het object zelf.

3.5.2 Syntax ADO.NET
Voor het realiseren van een connectie naar een database is het ADO.NET Connection object nodig. Met een ADO.NET Command object kan vervolgens data geselecteerd en/of gewijzigd worden. Een Command object wordt gebruikt voor het uitvoeren van SQL statements, zoals SELECT, INSERT, UPDATE, of DELETE statements, stored procedures, of ieder andere statement die geldig is voor de specifieke database.

```
Private Sub ListLoad()
  Dim cmdProducten As SqlClient.SqlCommand
  Dim drProducten As SqlClient.SqlDataReader
  Dim sqlString As String
  Dim connectionString As String

  ' Build Connection String
  connectionString &= "Data Source=(local);"
  connectionString &= "Initial Catalog=Northwind;"
  connectionString &= "User ID=sa"

  ' Build SQL String
  sqlString = "SELECT ProductName "
  sqlString &= "FROM Products"

  Try
    ' Create new Command object
    cmdProducten = New SqlClient.SqlCommand()
    With cmdProducten
      ' Create new Connection object
      .Connection = _
      New SqlClient.SqlConnection(connectionString)

      ' Open the connection
      .Connection.Open()
```

```
      ' Assign SQL statement to CommandText property
      .CommandText = sqlString

      ' Create DataReader object
      drProducten = .ExecuteReader()
    End With

    ' N.B. De volgende code hoort bij een n-tier applicatie
    ' thuis in de GUI. Voor het voorbeeld is het in dezelfde
    ' functie opgenomen.
    ' Clear the listbox
    lstProducts.Items.Clear()

    ' Start looping through the records
    ' The Read method returns false if it hits the end
    ' of the rows of data
    Do While drProducten.Read()
      ' Assign data to listbox
      ' Data returned from the Item property is of Object
      ' data type which is also required by the Add method
      ' the Items collection of the listbox. Therefore, no
      ' conversion is required.
      lstProducts.Items.Add(drProducten.Item("ProductName"))
    Loop

    Catch exp As Exception
      MessageBox.Show(exp.Message)

    End Try

End Sub
```

Het object Dataset heeft als functie een client-side cache te creëren van een of meer gerelateerde Recordsets. Een DataSet bevat één of meer DataTables die corresponderen met een databasetabel of view. De data in een DataSet kan worden weggeschreven of worden ingelezen als XML. Behalve de data in de tabellen kan een DataSet ook informatie bevatten over constraints en relaties tussen de data in de tabellen. Dit maakt het mogelijk om behalve sequentieel ook hiërarchisch te navigeren door de data in de DataSet.
De DataAdapter component heeft als functie alle datauitwisseling met een onderliggende database te verzorgen. Zo wordt met behulp van een DataAdapter een DataTable of DataSet gevuld met data. De onderstaande code is een voorbeeld van het creëren van een DataSet.

```
Private Sub DataSetCreate()
Dim northwindAdapter As SqlClient.SqlDataAdapter
Dim sqlString As String
Dim connectionString As String

  ' Build Connection String
  connectionString &= "Data Source=(local);"
  connectionString &= "Initial Catalog=Northwind;"
  connectionString &= "User ID=sa"

  ' Build SQL String
  strSQL = "SELECT * " sqlString &= "FROM Products"
  dsProducten = New DataSet ()

  Try
    ' Create New Data Adapter
    NorthwindAdapter = New _
   SqlClient.SqlDataAdapter(sqlString, connectionString)

    ' Fill DataSet From Adapter and give it a name
    northwindAdapter.Fill(dsProducten , "Producten")

    With dsProducten.Tables("Producten")
      .PrimaryKey = New DataColumn() _
      (Columns("ProductID")
    End With

  Catch exp As Exception
    MessageBox.Show(exp.Message)

  End Try

End Sub
```

3.5.3 Oracle Data Provider

Op dit moment is er een Data Provider voor Oracle te downloaden van http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/940/msdncompositedoc.xml. Oracle 8i Release 3 (8.1.7) Client of hoger moet geïnstalleerd zijn. De ‘System.Data.OracleClient’ namespace wordt automatisch aan de Global Assembly Cache toegevoegd. Doordat de Oracle Data Provider een Native Provider is, zal er onder water niet meer gebruik worden gemaakt van OleDb of ODBC providers. Hierdoor biedt de Oracle Data Provider grote performance voordelen ten opzichte van de OleDb Data Provider.

In Visual Studio .NET kunnen de Oracle componenten worden toegevoegd op de Data tab van de toolbox. Klik hiervoor met de rechtermuis op dit tabblad en kies ‘Customize toolbox…’. Op het tabblad ‘.NET Framework Components’ kunnen ‘OracleCommand’, ‘OracleConnection’ en ‘OracleDataAdapter’ worden geselecteerd zodat deze objecten vervolgens in Design mode gedefinieerd kunnen worden.

De connectstring voor een Oracle database kan er als volgt uit zien:

```
Data Source = mnpbase.rivm.nl;user id=userx;password=xxx
```

Waarschijnlijk bevat het .NET Framework bij de volgende versie standaard de Data Provider voor Oracle.

3.6 Ontwerp richtlijnen

Voor het bouwen van overzichtelijke, onderhoudbare applicaties is het van belang dat alle ontwikkelaars dezelfde manier van programmeren hanteren. Een volledig zelfde manier is een utopie aangezien elke ontwikkelaar zijn eigen niveau en visie heeft. Toch kan door eenvoudige afspraken duidelijke code ontstaan. Dit kan door gebruik te maken van naamconventies, commentaarheaders en commentaarregels. Gebruik bovendien voor de naam van de variabelen en controls een duidelijke uitgeschreven naam.
Microsoft hanteert ook voor haar eigen code strakke richtlijnen. Hierdoor ziet de code er in het Framework eenduidig uit. Het verdient dan ook de aanbeveling om dezelfde stijl van programmeren te hanteren, omdat veel code uit het framework gebruikt wordt.
In de volgende paragraaf worden de richtlijnen van Microsoft beschreven (in de literatuurlijst is meer te vinden over de Design Guidelines van Microsoft). Overigens heeft Microsoft een gratis tool gemaakt, genaamd FXCop, waarmee ontwerp richtlijnen gecontroleerd worden. Dit varieert van het juist gebruik van scope (Public, Friend, Private) tot de juiste naam conventies. Het mooie van deze tool is dat er ook zelf ontwerp richtlijnen aan toegevoegd kunnen worden en per project bewaard kunnen worden. Hierdoor is het controleren van een project zeer eenvoudig. Het is dan ook aan te raden om deze tool te gebruiken. De tool kan gevonden worden op: http://www.gotdotnet.com/team/libraries/

3.6.1 Naam conventie

VS.NET maakt gebruik van Strong Types. Dit wil zeggen dat .NET altijd controleert of de Types overeen komen. Op het moment dat een variabele van het type Long in een Integer geplaatst wordt, zal de ontwikkelaar door de IDE van .NET hierop gewezen worden. De code zal dan niet compileren. Verder kunnen verschillende talen binnen een VS.NET project door elkaar gebruikt worden. Het type ‘Single’ in VB.NET is in C# een ‘Float’, beide erven van het type System.Single uit het Framework (zie *Tabel. 3*). De prefix ‘int’ zegt voor een C# ontwikkelaar dus niets. Hierom is Microsoft afgestapt van de zogenaamde ‘Hongarian notation’. Er wordt nu van de Pascal- en camelCasing naam conventie gebruik gemaakt. DE PascalCasing notation begint altijd met een hoofdletter en elk nieuw woord in een naam begint ook met een hoofdletter. De camelCasing notation begint altijd met een kleine letter en elk nieuw woord in een naam begint met een hoofdletter. Daarnaast beginnen member variabelen met een ‘underscore’ en daarna camelCasing.

Tabel. 5: Naamgeving conventie

	naam conventie	voorbeeld
Namespaces	PascalCasing	PilotComponents
Public Objects	PascalCasing	Producten
Public Methods	PascalCasing	LaadScenarios
Private Methods	PascalCasing	BerekenEffectiviteit
Parameters	camelCasing	Bewaren(scenarioCode)
Private Class Members	_camelCasing	_kosten
Private Members	camelCasing	scenarioCode
Constants	UPPER_CASE	ERROR_MESSAGE

Verder heeft Microsoft nog een aantal richtlijnen voor het gebruik van properties, methods en parameters. Dit zijn:

- Gebruik geen Public Class Members, maar gebruik hiervoor Properties.
  Binnen een property is het namelijk mogelijk om de waarde te valideren en eventueel fouten af te vangen.
- Gebruik geen Write Only Properties, maar gebruik hiervoor een Method.
- Gebruik properties alleen om gegevens in weg te schrijven en uit te lezen.
  Zorg er voor dat de waarde die uit een property gelezen wordt dezelfde waarde heeft als de waarde die er is ingestopt. Met andere woorden, voer geen berekeningen e.d. uit in de Get.
- Zorg ervoor dat properties in een willekeurige volgorde ingevuld kunnen worden.
  Properties behoren niet van elkaars bestaan te weten. Met behulp van Methods kunnen gegevens gecombineerd worden.
- Gebruik dezelfde namen en volgorde bij parameters.
  Indien een functie gebruik maakt van overloading is het verstandig om dezelfde namen en volgorde voor de parameters aan te houden. Voor de consistentie is het ook verstandig om dezelfde naam voor een parameter aan te houden bij verschillende methods. (bijv. de parameter connectionString).

3.6.2 Control prefixes

Voor controls die gebruikt worden op zowel een Windows Form als een Web Form worden nog wel steeds prefixes aanbevolen.

Control	Prefix
Animation button	ani
Combobox	cbo
Checkbox	chk
Collection	col
(Command) Button	btn *(let op cmd wordt veel gebruikt voor het ADO command object)*
Common dialog control	dlg
Frame	fra
Graph	gra *(gph wordt ook veel gebruikt)*
Grid	grd
Horizontal scroll bar	hsb
Image	img
Label	lbl
Line	lin
Listbox	lst
Listview	lvw
Menu	mnu
Option button	opt
OLE control	ole
Picture	pic
Progressbar	prg
Report	rpt
Shape control	shp
Slider	sld
Spin control	spn
Statusbar	sta *(stb wordt ook veel gebruikt)*
Tabstrip	tab
Textbox	txt
Timer	tmr

Toolbar	tlb
Treeview	tre *(trv wordt ook veel gebruikt)*
Validation control	val
Vertical scroll bar	vsb

3.6.3 Data Access Objects (DAO)

Object	prefix
Connection	cnn
Container	con
Database	dbs *(db wordt ook veel gebruikt)*
DbEngine	dbe
Document	doc
Field	fld
Group	grp
Index	idx
Parameter	prm
Property	prp
QueryDef	qdf
Recordset	rst
Relation	rel
Report	rpt
Table	tbl
TableDef	tdf
User	usr
Workspace	wrk

3.6.4 ActiveX Data Objects (ADO)

Object	Prefix
Connection	cnn *(cnt wordt ook veel gebruikt)*
Command	cmd
Error	err
Field	fld
Parameter	prm
Recordset	rst

3.6.5 eXtended Markup Language (XML)

Object	Prefix
Attribute	att
Document	doc
Element	elm
Node	nod
Schema	xsd

3.6.6 Commentaarheader

Boven elke class kan met behulp van een commentaarheader meer informatie gegeven worden over het gebruik van de class, de scope, de belangrijkste methods en eventueel relaties tot andere classes. In C# bestaat een functie die alle commentaar uit een project kan wegschrijven naar een document. Hiervoor moet alle commentaar met drie slashes in plaats

van twee slashes beginnen (///). NB de twee slashes zijn het equivalent van de enkele quote (‘) die in VB voor commentaar gebruikt wordt. De verwachting is dat dit in VB.NET ook zal komen. Op die manier is zowel de code beschreven voor de ontwikkelaar die onderhoud moet plegen, maar is ook gemakkelijk documentatie te maken.
Een commentaarheader kan er als volgt uitzien:

```
'_____________________________________________________________
'
'Object           :  Instrumenten(Optional ByVal caseId)
'Parameters       :  caseId; voor instrumenten behorende bij
                    opgegeven case
'Instancing       :  MultiUse
'Edit             :  Readonly
'Collections      :  Instrument met Key Cstr(instrumentId)
'Omschrijving     :  De instrumenten wordt geladen met eventueel
'                    een caseId key. Aan de hand van deze sleutel
'                    wordt de collectie van instrument classes
'                    gevuld waarbij elk instrument class een
'                    record uit de Tbl_Instrumenten
'                    vertegenwoordigd.
'_____________________________________________________________
```

Ook bij lange of belangrijke functies kan het handig zijn om een commentaarheader op te nemen.

```
'_____________________________________________________________
'
'Functie          :  Load(Optional ByVal caseId) As Boolean
'Parameters       :  caseId; laad alleen opgegeven case
'Scope            :  Public Function
'Omschrijving     :  Alle cases worden vanuit de database
'                    ingelezen. Eventueel kan een caseId
'                    opgegeven worden, indien slechts één case
'                    opgehaald hoeft te worden. Indien het
'                    inladen mislukt geeft de functie False
'                    terug, indien alle gegevens opgehaald zijn
'                    wordt True terug gegeven.
'_____________________________________________________________
```

Tot slot kan het opnemen van zogenaamde ‘Regions’ de code ook overzichtelijker gemaakt worden. Een region bestaat uit een korte omschrijving van de code hieronder. Hierdoor kunnen alle verschillende soorten code, zoals properties, public functies en sub, private functies en subs, modale declaratie, api’s etc. gegroepeerd worden. Dit ‘dwingt‘ de ontwikkelaar om code te groeperen en vergemakkelijkt de beheerder bij het zoeken naar code. Bij grote objecten kan het handig zijn om de properties en functies onderling op alfabet te sorteren.
Een groot voordeel van regions is dat ze in de IDE van .NET in- en uitgeklapt kunnen worden. Hiervoor moet gebruik gemaakt worden van de code `#Region “naam”` en `#End Region`.

Een indeling van een class zou er als volgt uit kunnen zien.

```
#Region "  *** PRIVATE VARIABLES ***
  'Hier de code
#End Region

#Region "  *** PUBLIC PROPERTIES ***
  'Hier de code
#End Region

#Region "  *** PUBLIC METHODS ***
  'Hier de code
#End Region

#Region "  *** HELPER FUNCTIONS ***
  'Hier de code
#End Region

#Region "  *** CLASS METHODS ***
  'Hier de code
#End Region
```

Een ander voorbeeld van een indeling voor een formulier is als volgt.

```
#Region "  *** PRIVATE VARIABLES ***
  'Hier de code
#End Region

#Region "  *** PUBLIC METHODS ***
  'Hier de code
#End Region

#Region "  *** HELPER FUNCTIONS ***
  'Hier de code
#End Region

#Region "  *** FORM METHODS ***
  'Hier de code
#End Region

#Region "  ***  CONTROL EVENTS ***
  'Hier de code
#End Region
```

3.6.7 Commentaarregel

Tijdens het ontwikkelen van een applicatie ontstaan vaak afspraken, uitzonderingen en voorwaarden. Ondanks dat deze vaak gedocumenteerd worden in technische documentatie of een gebruikershandleiding is dit moeilijk terug te lezen in de code. Het gevaar bestaat dan dat er om een uitzondering heen een nieuwe uitzondering, met nieuwe voorwaarden wordt geprogrammeerd. Een onleesbare brei van If's, Loop's Select Case's is dan aan het ontstaan. Dit is voor de ontwikkelaar tijdens de bouw goed te begrijpen, maar niet meer voor een andere ontwikkelaar, of voor het bouwen van een nieuwe release na een jaar.

Bovenstaande is eenvoudig op te lossen door het toevoegen van commentaar regels. Zet deze regels voordat een If, For, Do of Select Case begint en leg in een enkele regel uit wat er gaat

gebeuren. Vaak helpt het om dit te herhalen bij de Else. Ook in het kort de If herhalen bij het End If statement wil verhelderend helpen, bij het lezen van de code.

```
Public Sub Form_Load()

'Indien geen Case opgegeven, dit melden in Caption form
  If Me.CaseID = 0 Then

    Me.Text = "Geen case geslecteerd."

  Else ' Titel instellen voor de geselecteerde Case

    Me.Caption = "Huidige case = " & Me.CaseID

    'Vervolgens de cboInstrumenten vullen
    For Each Instrument In Instrumenten

      cboInstrumenten.AddItem Instrument.Naam

    Next

  End If 'Me.CaseID = 0

End Sub
```

3.6.8 Enumeratie

Het gebruik van enumeratie kan voor zowel de ontwikkelaar zelf, als voor de beheerder van de applicatie erg handig zijn. Bovendien dwingt het gebruik van een enumeratie in bijvoorbeeld een parameter van een functie een juiste waarde voor de desbetreffende parameter af. De naamconventie voor een enumeratie is : NaamEnum (zie in voorbeeld TabbladEnum).

```
'CODE IN CLASSE
Sub Main()
  InstrumentenScherm.Tabblad = TabbladBewerken
  InstrumentenScherm.Show
End Sub

'CODE IN FORM frmInstrumenten

Public Enum TabbladEnum
  defaultTabblad = 0 'Selecteert het default tabblad
  tabbladInvoer
  tabbladBewerken
  tabbladVerwijderen
End Enum

Private _tabblad As TabbladEnum

Public Property Get Tabblad() As TabbladEnum
  Tabblad = _tabblad
End Property

Public Property Let Tabblad(tabblad As TabbladEnum)
  _tabblad = tabblad
End Property
```

```
Public Sub Form_Load()

  'Indien DefaultTabblad geslecteerd is dan niets instellen,
  'anders het opgegeven tabblad als eerste instellen.
  If Me.Tabblad <> defaultTabblad Then
    Me.tabInstrumenten.Tab = Me.Tabblad
  End If

End Sub
```

4. Architectuur

4.1 Infrastructuur

De structuur van de applicatie wordt vaak al bepaald tijdens de informatie analyse. Het is dan ook van belang om een goede keuze te maken in de te gebruiken structuur. In deze paragraaf wordt een structuur besproken die een grote flexibiliteit biedt tijdens en na de bouw. Hierdoor blijft de applicatie onderhoudbaar en is het gemakkelijker om hernieuwde inzichten van de opdrachtgever te kunnen verwerken. In dit hoofdstuk beschreven structuur is zeer geschikt voor grote applicaties (Enterprise applicaties). Het blijft daarom van belang om te bekijken of de hierna beschreven structuur toegepast kan worden in de gewenste applicatie.

In onderstaande afbeelding wordt de structuur van een .NET Enterprise applicatie weergegeven. Met een Enterprise applicatie wordt een applicatie bedoeld die op de Desktop PC van de gebruiker draait, maar die gebruik maakt van DLL's of Web Services op andere machines, met daaraan gekoppeld weer een Database server. Uiteraard kan in een Enterprise applicatie de User Interface ook uit een web interface bestaan.

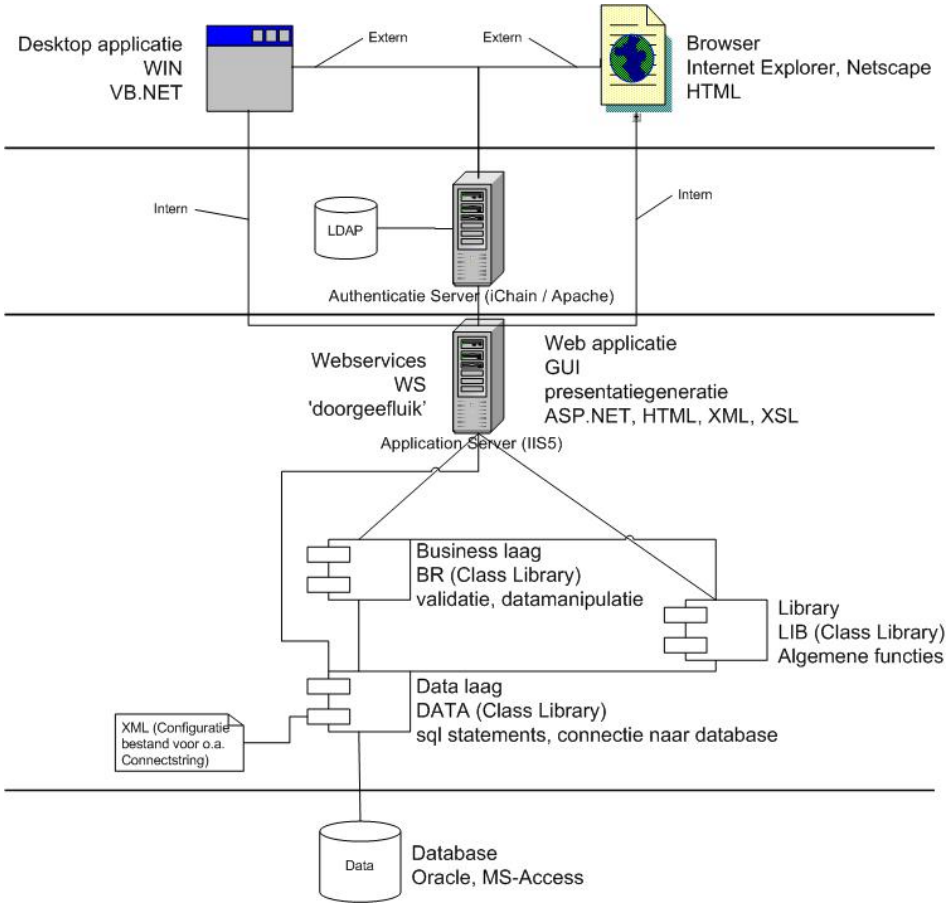


Fig. 7: Voorbeeld infrastructuur .NET Web applicatie, zoals gebouwd tijdens de pilot

Hierin zijn een viertal fysieke lagen te onderscheiden welke met een horizontale lijn worden weergegeven. De fysieke lagen zelf kunnen weer uit verschillende functionele lagen bestaan. Deze worden als een los object getoond.

Van boven naar beneden gezien kunnen de volgende lagen onderscheiden worden:

- Graphical User Interface (Desktop) / Webbrowser (functioneel en fysiek)
- Authenticatie Server (fysiek)
- Web Services (Desktop) / Graphical User Interface (Web) (functioneel en fysiek)
- Business laag (functioneel)
- Data laag (functioneel)
- Database server (functioneel en fysiek)

Verder is in de figuur nog een Library weergegeven. Dit is een verzameling algemene code die zowel door de GUI, de business laag als de data laag aangesproken kan worden. Hierbij kan gedacht worden aan stringmanipulatie, datum functies etc. Verder is in de figuur een XML configuratie bestand opgenomen. Met dit bestand is het mogelijk om de applicatie eenvoudig aan een andere database server te koppelen.

Door een applicatie op te delen in lagen kan deze eenvoudig in de MNP ICT Infrastructuur (zoals beschreven in de A&K analyse) gebruikt worden. De scheiding van User Interface en Business logica kan namelijk ook fysiek worden doorgevoerd door beide lagen op een verschillende server te plaatsen. Ook is het mogelijk om de authenticatie door de standaard RIVM servers te laten afvangen.

4.2 Client

De eerste laag bestaat uit de client PC. Op de client wordt de User Interface van de applicatie getoond. Indien gebruik wordt gemaakt van een VB.NET desktop applicatie moet op elke client het .NET Framework geïnstalleerd worden. De client zal immers de applicatie Just In Time compileren naar Native Code. Bij een Web applicatie heeft de gebruiker slechts een browser nodig. Aangeraden wordt om gebruik te maken van Microsoft Internet Explorer 6. Deze versie van Internet Explorer ondersteunt .NET Web applicaties het beste.
Bij een Desktop applicatie ligt de User Interface opslagen in de executable op de client. Bij een Web applicatie wordt de User Interface opgehaald van de web server via het HTTP protocol.

4.3 Authenticatie server

Bij het RIVM wordt eventueel een tweede fysieke laag toegepast als beveiliging voor de applicatie. Dit geldt altijd wanneer de client zich buiten de organisatie bevindt. De Authenticatie Server bestaat uit een iChain Proxy Server draaiend op een Apache Webserver. De Proxy Server valideert de gebruiker tegen een LDAP Server (Oracle). Indien de gebruiker bekend is, dan wordt de gebruiker doorgesluisd naar de juiste Applicatie server met behulp van Reverse Proxy.

4.4 Applicatie server

De applicatie server is de derde fysieke laag in de figuur in paragraaf 4.1. De applicatie server draait op een Internet Information Service Web Server. Op de applicatie server staan drie lagen die zich functioneel onderscheiden, te weten Web Services, Business laag en Data laag voor Desktop applicaties, en User Interface laag, business laag en Data laag voor Web applicaties. Op de applicatie server is in beide gevallen het .NET Framework noodzakelijk. In het ene geval om de Web Services te kunnen interpreteren, in het andere geval om de ASPX pagina’s te interpreteren.

4.4.1 Web Services

De Web Services technologie is bedoeld als vervanging van DCOM voor communicatie tussen verschillende PC's. Web Services worden gebruikt op een web server en kunnen door elk willekeurig device worden aangesproken. Binnen een Web Service worden vaak weer referenties gelegd naar DLL's die de logica bevatten die uitgevoerd moet worden. Doordat een Web Service vooral bedoeld is om remote aanspreken mogelijk te maken, is het vaak niet meer dan een doorgeefluik van data.

Een Web Service stelt programmeerbare applicatielogica ter beschikking en is zelf toegankelijk via standaard internetprotocollen en gegevensindelingen, zoals HyperText Transfer Protocol (HTTP) en eXtensible Markup Language (XML). De extensie van een Web Service is ASMX. Bovendien wordt een Web Service-interface strikt gedefinieerd door de berichten die de Web Service accepteert en genereert. Hierover zijn internationale afspraken gemaakt die gewaarborgd worden door het World Wide Web Consortium (W3C). Deze afspraken hebben tot doel om de Internet technologieën van verschillende bedrijven zo goed mogelijk op elkaar te laten aansluiten. Indien een Web Service wordt aangesproken zal deze data in XML formaat terug geven. In de volgende figuur wordt de verkregen XML rechtstreeks in de browser getoond.

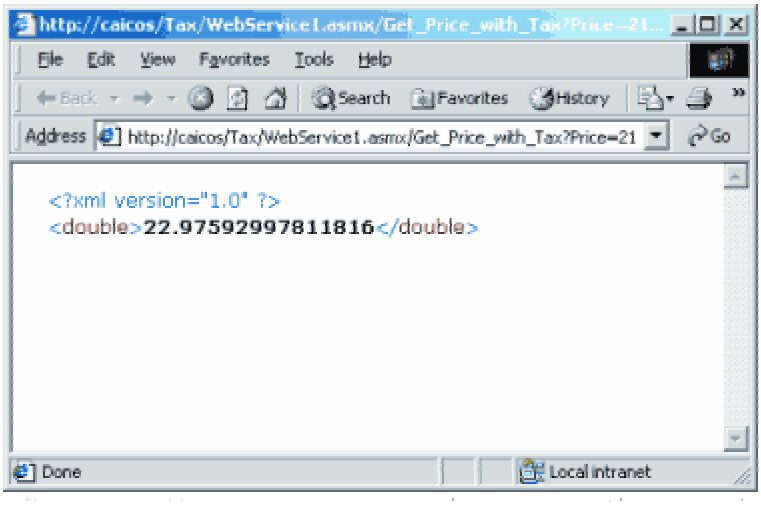


Fig. 8: XML resultaat van een Web Service

UDDI

Ontwikkelaars hebben ook een manier nodig om Web Services op te sporen. De specificatie van het Discovery Protocol (Disco) definieert een indeling voor het discovery document (gebaseerd op XML) en een protocol voor het binnenhalen van het discovery document, zodat ontwikkelaars services op een bekende URL kunnen opsporen. Vaak zal de ontwikkelaar de URL's waar services gevonden kunnen worden echter niet kennen. Universal Description, Discovery and Integration (UDDI) specificeert een mechanisme waarmee aanbieders van Web Services het bestaan van hun services bekend kunnen maken en waarmee gebruikers dergelijke services kunnen opsporen. De UDDI-specificatie bestaat uit drie delen:

- White pages die contactinformatie over bedrijven bieden;
- Yellow pages die Web Services indelen in categorieën (zoals 'Creditcardverificatie');
- Green pages die gedetailleerde technische informatie over afzonderlijke services bieden.

Het UDDI Business Registry is een implementatie van de UDDI-specificatie, die op zijn beurt zelf een Web Service is die SOAP via HTTP gebruikt als berichtenprotocol.

WSDL
De Web Services Description Language (WSDL) voorziet in een standaard manier om te beschrijven welke functies een specifieke Web Service biedt en welke argumenten moeten worden doorgegeven om deze functies aan te roepen. WSDL is, net als SOAP, voorgedragen bij het W3C voor standaardisatie.

SOAP
Het Simple Object Access Protocol (SOAP) is gebaseerd op XML. Hiermee worden toepassingen in staat gesteld elkaar aan te roepen op een standaard, flexibel gekoppelde wijze, waardoor het mogelijk wordt gedistribueerde toepassingen te bouwen die werken via het Internet. In het geval van Web Services wordt met behulp van SOAP beschreven hoe een Web Service aangeroepen moet worden. De inhoud van zo'n aanroep gebeurt met XML.

Een SOAP bericht bestaat uit de volgende onderdelen:
- Envelop: definieert een framework voor het beschrijven van wat in een bericht staat en hoe het te verwerken;
- Header: bevat additionele informatie (vaak informatie die niet strict noodzakelijk is voor de uit te voeren operatie), is optioneel;
- Body: een conventie voor de representatie van 'remote procedure calls' en antwoorden.

SOAP definieert een standaard XML berichtstructuur. Een SOAP bericht kan worden gezien als een envelop. Op de envelop staat het adres waarnaar het bericht moet worden verstuurd: de header van een SOAP bericht. In de header staat de URL. De envelop zelf bevat het werkelijke bericht. Dit bericht bestaat onder meer uit de functionaliteit die wordt aangeroepen, alsmede de parameter(s) die vereist zijn om de functionaliteit uit te kunnen voeren. SOAP maakt gebruik van XML namespaces die tags voor de structuur van de envelop en voor de codering van de gegevens definiëren.

Zoals in onderstaande afbeelding is weergegeven kan een SOAP bericht meerdere headers en bodies bevatten.

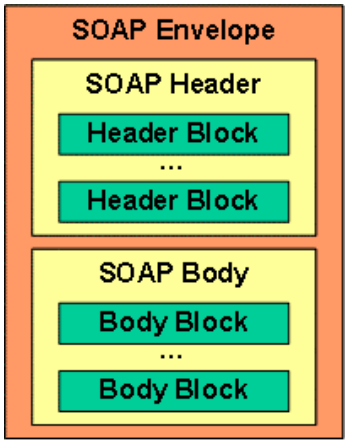


Fig. 9: Opbouw SOAP bericht

SOAP is onafhankelijkheid van taal, technologie, platform, en technische implementatie. Naast HTTP kunnen in de huidige SOAP specificatie (versie 1.1) ook andere protocollen als MSMQ, MQ Series, SMTP en FTP/IP het transport van de boodschap vervullen.

Hieronder wordt een voorbeeld gegeven van een SOAP bericht voor een Web Service die een melding genereert.

```
<?xml version="1.0" encoding="UTF-8" ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope">
<env:Header>
<n:alertcontrol xmlns:n="http://example.org/alertcontrol">
<n:priority>1</n:priority>
<n:expires>2002-12-22T11:00:00-12:00</n:expires>
</n:alertcontrol>
</env:Header>
<env:Body>
<m:alert xmlns:m="http://example.org/alert">
<m:msg>Afdelingsoverleg 12 december om 11:00 uur.</m:msg>
</m:alert>
</env:Body>
</env:Envelope>
```

4.4.2 Web GUI

Bij een Web applicatie staat de code voor de User Interface op de applicatie server. Dit kan bestaan uit HTML pagina's, maar ook uit in ASP.NET geschreven ASPX pagina's. Een aanvraag van een client wordt verwerkt door Internet Information Service (IIS) op de web server. Als het om een HTML of HTM pagina gaat wordt de HTML code naar de client gestuurd. Bij een ASPX pagina wordt de code op de server geïnterpreteerd en uitgevoerd en vervolgens wordt de gegenereerde HTML code naar de client gestuurd.

4.4.3 Business laag

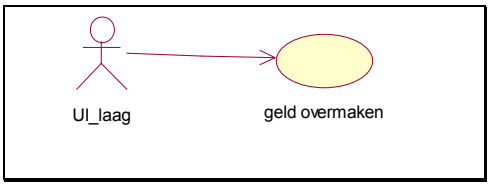
Verder bevindt op de applicatie server ook de business laag. De business laag bevat de logica van de applicatie. Hierin wordt dus de code geplaatst die de daadwerkelijke functionaliteit van de applicatie bevat. Hierbij wordt gebruik gemaakt van Classes. Om Performance redenen kan er wel gekozen worden om de Business laag en of de Data laag op een aparte server te draaien. Indien mogelijk is het echter aan te raden om de vierde laag te draaien op de applicatie server (web server van de derde laag). Dit voorkomt dat de vierde laag aangesproken moet worden via DCOM, .NET Remoting, of via een Web Service.

Binnen de business laag kan onderscheid gemaakt worden tussen verschillende classes.

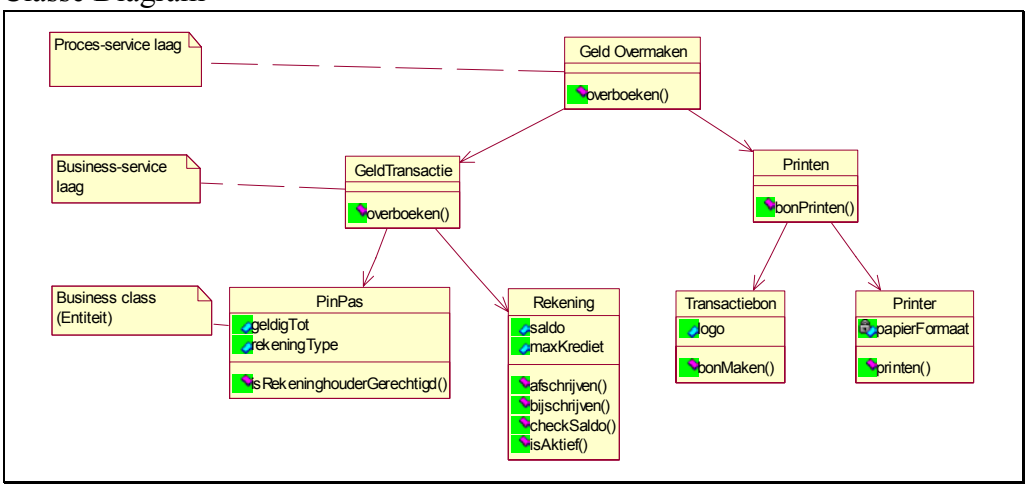
- Business class (entiteit): zorgt voor integriteit binnen het object. Dit object kan ook properties hebben. Een Business class weet niets van andere objecten.
- Business service: vormt een actie. Een actie kan meerdere Business classes nodig hebben. Binnen de actie wordt dan instanties gemaakt van deze Business classes. Dit vormt ook de relatie tussen de Business classes, aangezien een Business class niet van het bestaan weet van andere classes.
- Process service: voert meerdere acties uit (proces). Een Process service kan meerdere Business services aanspreken.

Om de verschillen in de typen classes te verduidelijken volgt hierna een voorbeeld van een bank applicatie waarbij iemand geld overmaakt van zijn rekening naar een andere rekening. Ter verduidelijking is het voorbeeld in UML weergegeven.

Use Case



Classe Diagram



Sequence Diagram

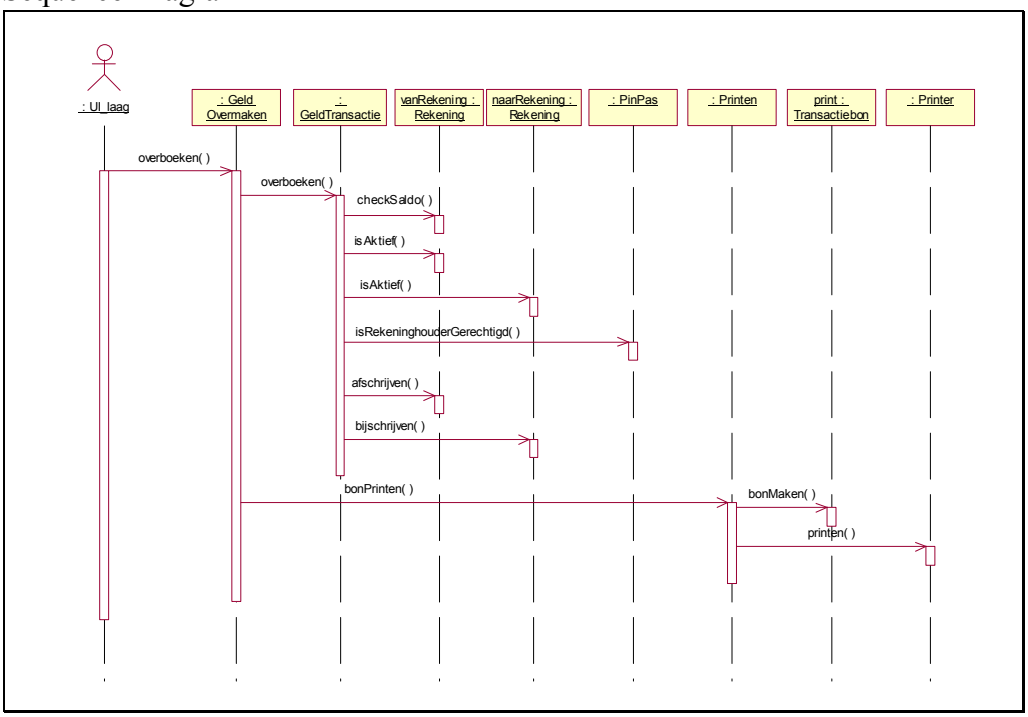


Fig. 10: UML weergave verschillende object types

De workflow van de code zal er ongeveer als volgt uit zien:
In de user interface zal er ingevuld worden hoeveel geld er over gemaakt dient te worden en de ‘van’ en ‘naar’ rekening nummers. Vervolgens zal de klant zijn pinpas invoeren.
Er wordt een Proces service ‘Geld overmaken’ gestart. Dit proces roept een Business service ‘Geld Transactie’ aan. De Business service maakt vervolgens twee instanties van de Business class ‘Rekening’ aan. Eén voor de ‘van’ rekening en één voor de ’naar’ rekening. Bij de van

rekening wordt gevalideerd of er nog genoeg saldo aanwezig is en of de rekening nog actief is. Bij de naar rekening wordt gevalideerd of de rekening nog actief is. Vervolgens wordt bij de Business class ‘Pinpas’ gevalideerd of de rekeninghouder gerechtigd is om van de opgegeven rekening geld af te halen. De Business class ‘Rekening’ en ‘Pinpas’ weten dus niets van elkaar. Dit loopt via de Business service ‘Geld transactie’. Wanneer alles gevalideerd is zal het geld overgemaakt worden. Tot slot zal het Proces ‘Geld overmaken’ nog een instantie van de Business service ‘Printen’ instantieren. Deze Business service zal vervolgens de Business class ‘Transactiebon’ en ‘Printer’ instantieren en de gegevens van de Business class ‘Transactiebon’ door de Business class ‘Printer’ naar de printer laten sturen. Het complete proces is nu uitgevoerd. Het geld is overgemaakt en de klant krijgt als bewijs een transactiebon mee.

NB. In het geval van een Desktop applicatie wordt de business laag soms nog weer gesplitst in een ‘Business Rules’ laag en een ‘Business object’ laag. De ‘Business Rules’ laag bevat de eenvoudigere validatie van de objecten, maar heeft wel dezelfde structuur als de business object laag. Deze laag bevindt zich samen met de User Interface op de client. Deze structuur wordt vaak gekozen om performance redenen door eenvoudige validaties op de clients uit te kunnen voeren en eventueel gebruik te kunnen maken van statefull objecten.

4.4.4 Library project

Soms wordt gebruik gemaakt van een zogenaamde Library project. Hierin staan algemene functies die zowel door de GUI, de business laag als de data laag gebruikt worden. Hiebij kan gedacht worden aan String manipulatie, datum manipulatie etc. Indien deze library gebruikt wordt door meerdere projecten die ook nog eens op verschillende web servers staan, dan is het aan te raden om hiervoor een Web Service te bouwen. Deze kan dan weer vrij eenvoudig met behulp van een Web Reference gekoppeld worden aan het project.

4.4.5 Data laag

Op de applicatie server staat ook de Data laag. Deze bestaat uit Classes en een Component Class Object die een connectie maakt naar een database (Oracle, SQL Server, MS Access etc.) om gegevens op te halen, te wijzigen of te verwijderen. Vanuit de Business laag kan een referentie gemaakt worden naar de Data laag. Door de data laag logisch te scheiden en niet Business laag zelf connectie te laten maken naar de database, onstaat een onderhoudbare applicatie. Alle SQL statements staan bij elkaar in de Data laag, en het is mogelijk de Businesslaag eenvoudig naar een andere Data laag te laten wijzen.

4.4.6 Factory Class

De Factory class is een andere naam voor een zelf geschreven component dat ervoor zorgt dat de datalaag database onafhankelijk wordt. Let wel, het is geen vervanging van ADO.NET, maar maakt juist gebruik van één van de mogelijkheden van het .NET Framework. Zoals beschreven in paragraaf 3.5 bestaat ADO.NET uit onder andere het DataAdapter, Command en Connection object. Standaard in VS.NET zijn deze objecten er in drie varianten, te weten ODBC, OLEDB en SQL. De laatste is specifiek ontwikkeld voor SQL Server, de eerste twee voor alle overige databases.

De variant voor SQL wordt ook wel Native Provider genoemd. In 3.5.3 werd al aangegeven dat er inmiddels ook een Native Provider voor Oracle op de markt is gebracht. Op het moment van schrijven is er ook een Native Provider voor MySQL databases uitgekomen en waarschijnlijk zullen er nog een aantal andere Native Providers ontwikkeld worden. Een

groot voordeel van deze zogenaamde Native Providers is dat ze specifiek voor één database ontwikkeld zijn. Ze zijn daarom sneller, omdat ze onder water niet meer gebruik maken van OleDb of ODBC providers. En de SQL Server Native Provider heeft bijvoorbeeld als extra bijkomstigheid dat hij rechtstreeks uit de database kijkt naar de beschikbare tabellen, kolommen, etc. Deze informatie wordt onder andere gebruikt voor Intellisense in de IDE.

Alle objecten binnen de Native Providers erven van de algemene Interface van deze objecten. Zo erft bijvoorbeeld zowel het 'OleDbConnection' object als het 'SqlConnection' object van de Interface 'IDbConnection'. De (zelfgeschreven) Factory class maakt hier gebruik van. Binnen de applicatie wordt de Factory class aangeroepen die altijd een Interface object retourneert. Binnen de Factory class wordt afgevangen welke database op dat moment in gebruik is en wordt de juiste Native Provider gebruikt. Door het gebruik van een Interface wordt nog steeds gebruik gemaakt van ADO.NET.

```
Imports System.Data
Imports System.Data.Odbc
Imports System.Data.OleDb
Imports System.Data.OracleClient
Imports System.Data.SqlClient

Public Class FactoryClass
  Inherits System.ComponentModel.Component

  'Defineer de database types
  Private Enum databaseTypes
    dtAccess
    dtOdbc
    dtOleDb
    dtOracle
    dtSqlServer
  End Enum

  'Vul hier een database type in, of haal dit uit een
  'configuratie bestand.
  Private _databaseType As databaseTypes = _
  databaseTypes.dtOracle
  Private _connection As IDbConnection

  Public Function GetConnection() As IDbConnection
    Select Case _databaseType
      Case databaseTypes.dtOdbc
        _connection = New OdbcConnection()
      Case databaseTypes.dtOleDb, databaseTypes.dtAccess
        _connection = New OleDbConnection()
      Case databaseTypes.dtOracle
        _connection = New OracleConnection()
      Case databaseTypes.dtSqlServer
        _connection = New SqlConnection()
    End Select
  End Function

End Class
```

4.5 Database server

Tot slot is er nog één fysieke laag te weten de database server. Voor optimale scheiding van functionaliteit kan gekozen worden om gebruik te maken van Stored Procedures in plaats van SQL statements in de applicatie. De data laag kan met behulp van ADO.NET deze Stored Procedures aanspreken en daarmee hoeft de datalaag dus geen informatie te bevatten van de structuur van de database. De database server wordt in de meeste gevallen beheerd door een DBA en valt dan buiten de verantwoordelijkheid van de .NET ontwikkelaar.

5. User Interface

Bij het maken van een User Interface moet als eerste een keus gemaakt worden of de applicatie als een Desktop applicatie of als een Web applicatie gebruikt gaat worden. Zoals in hoofdstuk 4 te lezen is, heeft de keuze voor de User Interface invloed op de infrastructuur en de invulling van diverse lagen. Ook de manier waarop de User Interface gepresenteerd wordt aan de gebruiker verschilt bij deze twee typen.
Bij een Desktop applicatie is gezien de aanwezige VB6 ervaring de keus VB.NET en moeten er Windows Forms gemaakt worden. Overigens heeft de keus voor C# door C of Java ontwikkelaars geen invloed op het Type Forms. Ook zij zullen gebruik maken van de Class Library Windows Forms. Alleen zal de programmeertaal achter elk Form anders zijn.
Bij een Web applicatie is de keus ASP.NET en worden er Web Forms gemaakt voor de User Interface. De programmeertaal achter ASP.NET kan de ontwikkelaar zelf kiezen. Aangeraden wordt om hier dezelfde taal als bij de Desktop applicatie te kiezen. Om een betere keuze te kunnen maken tussen de twee typen User Interface wordt er in dit hoofdstuk dieper ingegaan op dit onderdeel.

5.1 Verschillen Desktop en Web

Ondanks dat IT steeds meer Internet gericht wordt, blijven Desktop applicaties rijker aan lay-out mogelijkheden. Ook de mogelijkheid tot interactie met het werkstation, denk aan registry, common dialogs en messageboxes, is veel groter. Alleen al uit beveiligingsoogpunt is dit lastiger te realiseren voor Web Applicaties. Dit heeft duidelijk zijn invloed op de (on)mogelijkheden die een applicatie kan bieden. Even een Excel spreadsheet openen op de client om rekenwerk uit te voeren is er bij een Web applicatie bijvoorbeeld niet meer bij. Ook lokale Access databases behoren tot de verleden tijd bij een Web applicatie. Daarentegen zijn Web applicaties gemakkelijk te integreren in een bestaand Intranet en behoort de distributie langs vele clients tot de verleden tijd. De ontwikkelaar hoeft slechts de applicatie op de web server en applicatie server te zetten. Ook het uitrollen van testversies is op deze manier eenvoudig te realiseren.

5.1.1 Controls

Een groot verschil tussen Desktop en Web User Interfaces (lees het verschil tussen de ‘System.Forms’ en ‘System.Web.UI’ namespaces) is de rijkheid aan controls. Hierin is duidelijk te zien dat er op de Desktop veel meer mogelijkheden zijn. Toch hoeft dit geen belemmering te zijn om Web applicaties te bouwen. De meest gebruikte controls zoals ‘label’, ‘textbox’, ‘combobox’, ‘listbox’en ‘commandbutton’ zijn gewoon aanwezig, terwijl veel gebruikte controls zoals ‘treeview’, ‘menu’ en ‘tab’ control missen. Hiervoor zijn op het Internet wel zogenaamde 3rd party controls te vinden. De kwaliteit van deze controls is soms uitstekend, maar laat soms ook te wensen over. Hierdoor is veel ontwikkeltijd nodig om de functionaliteit van de controls na te bouwen.

5.1.2 Help

Voor het maken van help pagina’s wordt bij het MNP gebruik gemaakt van de nieuwste versie van RoboHelp (X3). Deze nieuwe versie ondersteund naast HTML help ook Web help. Bij HTML help wordt elke helppagina als een HTML pagina aangemaakt en achteraf samengevoegd tot één .chm bestand. Bij VB6 applicaties werd een .chm bestand gekoppeld aan de applicatie.

Bij Web help is elke helppagina nog steeds een HTML pagina en kan ook nog steeds één .chm bestand gegenereerd worden. Een uitbreiding van Webhelp is dat er voor de desktop applicaties een api- wordt meegeleverd voor context-sensitive help. Hierdoor is het genereren en de distributie van het .chm bestand overbodig.

ASP.NET help
Help is volledig geïntegreerd in ASP.NET. Via ASP.NET is het mogelijk om hyperlinks toe te voegen. Nadat op de webserver help is aangemaakt kunnen de help pagina's via hyperlinks gekoppeld worden aan de webforms.

VB.NET help
Help voor VB.NET kan op 2 manieren worden gebruikt.
- Op de traditionele manier met een .chm bestand
- Via de webserver.

Help op de traditionele manier met een .chm bestand werkt gelijk aan help in VB6 met Robohelp. Nadat de help pagina's zijn aangemaakt kan per scherm(element) onder properties de HelpContextID worden ingegeven. Dit is een willekeurig nummer dat later overeen komt met een *map#* in RoboHelp.

Het aanbieden van help via de webserver heeft als voordeel dat de help pagina's centraal (op de webserver) worden beheerd. Hierdoor hoeven de help pagina's niet gedistribueerd te worden. Ook is het achteraf nog mogelijk om de helpbestanden aan te passen terwijl de applicatie al gedistribueerd is.
Robohelp genereert hiervoor een html pagina genaamd 'project_rhc.htm'. In combinatie met een meegeleverde module (robohelp_csh.bas) kan hiermee context sensitive help aangemaakt worden. Na het toevoegen van Robohelp_csh.bas in het project is een API bruikbaar (RH_ShowHelpById) om specifieke help pagina's op te roepen.

```
Private Sub cmdHelp_Click()
  Dim window As RH_WindowOption 'RH_WindowOption is
  gedefinieerd in robohelp_csh.bas

  ' Definieer grootte helpWindow
  window.m_nLeft = 10
  window.m_nTop = 10
  window.m_nHeight = 500
  window.m_nWidth = 600
  window.m_nStyle = 0

  'Via internet op Windows2000 PC
  RH_ShowHelpById _
    "http://<ip-adres>/HELP/WebHelpInVb6_rhc.htm",_
     mapId,_
    "Overbodige parameter", _
     window

End Sub
```

5.2 Windows Forms

In het Framework valt de Class Library voor Windows Forms onder de namespace 'System.Forms'. Wat het eerst opvalt is de grote hoeveelheid aan controls die beschikbaar

zijn uit het .NET Framework voor Desktop applicaties. Hierdoor is het gebruik van API calls minder gauw noodzakelijk. Alle standaard common controls zijn bijvoorbeeld aanwezig. Hoewel Windows Forms een Class Library is geworden, komen ze verder veel overeen met de Forms in VB6. Daarom zal er in deze Reference Guide niet verder op worden ingegaan.

5.3 Web Forms

Web Forms lijken erg veel op Windows Forms. Net zoals in Visual Interdev is het mogelijk om Web pagina's met controls op te bouwen door middel van Drag-and-Drop. Een groot verschil is echter dat Web Forms nu in dezelfde IDE gebouwd kunnen worden als bijvoorbeeld een VB.NET DLL. Hierdoor kan er nu gemakkelijk gedebugged worden. De belangrijkste namespaces voor het bouwen van Web Forms zijn 'System.Web.UI' en 'System.Web.UI.Page'.

5.3.1 Client side / Server side controls

ASP.NET maakt het bouwen van Web applicaties gemakkelijker door de toevoeging van server controls waardoor minder zelfgeschreven code nodig is dan met ASP. Het tonen van data, valideren van gebruikersinvoer en uploaden van bestanden zijn voorbeelden hiervan. Daarnaast detecteert een server control welke type browser de client heeft en genereert hiervoor HTML code. ASP.NET pagina's werken in alle browsers waaronder Netscape, Opera, AOL, and Internet Explorer.

5.3.2 Session

Met behulp van het Session object (reeds bekend van ASP) kunnen gegevens in het geheugen van de web server worden opgeslagen. Dit kan handig zijn om op deze manier enige state van de applicatie vast te houden. Een goed voorbeeld waarvoor het Session object gebruikt kan worden is het opslaan van een Dataset. Stel, een combobox met teams wordt getoond op meerdere pagina's. Om te voorkomen dat bij het openen van deze pagina's elke keer de database moet worden aangesproken kan de gehele Dataset in het Session object geplaatst worden.
Hierbij moet echter wel goed uitgekeken worden. Indien het om een grote Dataset gaat (stel 400 kB), dan wordt bij honderd gebruikers al gauw enkele tientallen megabytes aan geheugen gebruikt van de web server. Het opnieuw aanspreken van de database is dan wellicht een snellere optie. Gebruik het Session object voor data opslag dan ook met mate.

Het Session object wordt op de volgende manier gevuld.

```
Session("sleutel") = "waarde"
```

Het Session object blijft gevuld totdat de browser gesloten wordt. Indien de gebruiker een nieuwe browser opent, dan zullen de gegevens in het Session object van de eerste browser niet beschikbaar zijn voor de tweede gebruiker. De web server houdt zelf met behulp van een Session ID bij, bij welke PC en welke browser sessie de gegevens horen.
Nieuw is dat de ASP.NET session state het mogelijk maakt om gebruikerspecifieke sessie informatie te delen over alle servers in een Web farm. Een Web farm is een combinatie van meerdere web servers. Het gaat echter te ver om de toepasbaarheid van Web farms hier uit te leggen.

5.3.3 Viewstate

Elke ASP.NET pagina heeft een zogenaamde viewstate. Web pagina's zijn stateless, maar met behulp van de viewstate wordt de inhoud van de controls opgeslagen. Dit zorgt ervoor dat bij een round-trip van de pagina naar de server alle controls weer worden gevuld met de waarden die de controls hadden. Onder een round-trip wordt verstaan dat de webserver wordt aangesproken en vervolgens dezelfde (bijgewerkte) pagina weer wordt getoond. Dit gebeurd bijvoorbeeld bij het klikken om een button. Bij zo'n round-trip worden alle control-events (click, change etc.) afgehandeld.

De viewstate is te bekijken in een hidden textbox in de HTML source. het heeft overigens geen zin om de viewstate op de client aan te passen, aangezien de viewstate door de server wordt afgehandeld. Het is wel mogelijk om in code de viewstate te vullen, c.q. te wijzigen.

```
<input type="hidden" name="__VIEWSTATE"
 value="dDwtMTM3MjE5ODEzNTs7PiE+6ENS4qHDhNK02FGLJf8vMcmI" />
```

Let wel op, het lijkt of de viewstate encrypted is en dat het daarom veilig is om er bijvoorbeeld een wachtwoord of andere gevoelige data in te stoppen. De viewstate is echter Base-64 encoded en kan ook gemakkelijk gedecodeerd worden. Het gebruik van de viewstate heeft invloed op de performance van de website. Doordat alle data opgeslagen wordt in de viewstate moet er bij grote pagina's, met bijvoorbeeld een datagrid, ook veel data verstuurd worden naar de client. Het kan in sommige gevallen dan ook raadzaam zijn te overwegen niet alle controls op te nemen in de viewstate.

Een goed voorbeeld van gebruik van de viewstate is de 'Passport' aanmeldpagina van Microsoft. De gebruiker vult zijn naamgegevens in en kiest vervolgens een land uit een combobox. Op het click-event van deze combobox wordt de pagina opnieuw getoond, maar nu uitgebreid met een combobox provincie, gevuld met de provincies van Nederland. De reeds eerder ingevulde naamgegevens zijn nu door de webserver opnieuw op de pagina ingevuld. De Viewstate bestaat zolang de pagina geopend is.
Het is ook mogelijk om zelf viewstate variabelen maken. Deze kunnen op nagenoeg dezelfde manier als het Session object gevuld worden.

5.3.4 Cookies

Met behulp van cookies kan 'state' informatie op de client worden weggeschreven en uitgelezen. Hierdoor kan in een webapplicatie informatie over bijvoorbeeld geselecteerde items en gebruikersvoorkeuren worden opgeslagen en op een later tijdstip weer worden gebruikt.

De Cookies collectie definieert een waarde voor een cookie (bijvoorbeeld Response.Cookies("Cookienaam")=waarde). Als de gespecificeerde cookie nog niet bestaat wordt deze aangemaakt. Bestaat de cookie wel, dan wordt de waarde overschreven. Met de Cookies collectie kunnen ook de waarden worden uitgelezen die in een HTTP request zijn meegestuurd (Request.Cookies("Cookienaam").Value).

Een cookie kan een 'expiration date' hebben. Als deze datum voorbij is wordt het cookie genegeerd. Bij een 'expiration date' met een datum in het verleden of de waarde 0 wordt een cookie direct genegeerd. Een cookie zonder 'expiration date' wordt niet naar de harde schijf van de gebruiker weggeschreven en is alleen geldig zolang de browser open is.

Bij het gebruik van cookies moet rekening worden gehouden met het beperkte aantal cookies die op een bepaald tijdstip aanwezig kunnen zijn:
- 20 cookies maximaal per domein
- 4096 bytes per cookie beschrijving
- 300 cookies maximaal in totaal

N.B. Internet Explorer heeft niet de limiet van 20 cookies per domein (Netscape wel).

Een gebruiker kan in de browser aangeven dat het niet toegestaan is dat cookies worden weggeschreven, maar standaard staat een ‘browser’ het wel toe. Door de lengte op te vragen van de server variabele ‘HTTP_Cookie’ kan worden bepaald of de browser cookies toestaat. Bij een lengte van 0 worden cookies geblokkeerd (zie onderstaande voorbeeld code). Indien de gebruiker heeft aangegeven dat de browser geen cookies toestaat, kan ervoor gekozen worden een ‘Session’ variabele te gebruiken i.p.v. een cookie.

```
Private Sub Page_Load(ByVal sender As System.Object, _
                      ByVal e As System.EventArgs) _
                      Handles MyBase.Load
  'Put user code to initialize the page here

Private Sub btnOpslaan_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
                             Handles btnOpslaan.Click

  If Len(Request.ServerVariables("http_cookie")) > 1 Then

    'Browser staat cookies toe
    If txtCookie.Text <> "" Then
      Response.Cookies("Tekst").Value = Me.txtCookie.Text
      Response.Cookies("Tekst").Expires = Today.AddDays(1)
      lblShowCookie.Text = Request.Cookies("Tekst").Value
      lblShowTextbox.Text = txtCookie.Text
    Else
      Me.lblShowCookie.Text = "Type tekst in en klik op Opslaan."
    End If

  Else

    'Cookies zijn niet enabled in de browser van de client
    Me.lblInformatie.Text = "Cookies zijn niet enabled " & _
                            "in uw browser!"
    Me.lblInformatie.Visible = True

  End If

End Sub
```

6. Hardware en Software

In dit hoofdstuk wordt gekeken naar de hardware en software die nodig is voor het ontwikkelen en gebruiken van .NET applicaties. Hierbij wordt onderscheid gemaakt tussen het ontwikkelen, het gebruiken en het hosten van .NET applicaties.

6.1 Ontwikkel PC

Ondanks dat VS.NET te installeren is op Windows 98 of hoger, werkt VS.NET pas goed op Windows 2000 of Windows XP. De versie van Windows 2000 (Professional, Server, Advanced Server) maakt niet veel uit. Voor het bouwen van Web applicaties voldoet een OS lager dan Windows 2000 zelfs helemaal niet. Het is mogelijk om gebruik te maken van Personal Web Server (PWS), maar deze interpreteert de ASP.NET code niet, of niet goed. Ook de Web Services zullen niet goed functioneren met PWS.
ASP.NET werkt wel goed met Internet Information Service (IIS). Dit dient wel minimaal versie 5.0 te zijn, maar deze staat standaard op een Windows 2000 PC.
Een Web applicatie kan in zijn geheel op een ontwikkel PC gebouwd worden, aangezien een ontwikkel PC beschikt over een web server. Toch is het verstandig om op tijd de applicatie ook op een (test) web server te installeren. Dit voorkomt onaangename verrassingen tijdens het opleveren van de applicatie. Bovendien stelt dit (een select groepje) gebruikers in staat om de applicatie te testen, terwijl de ontwikkelaar op zijn eigen PC kan verder ontwikkelen.

Wat gelijk opvalt is dat VS.NET veel meer geheugen vraagt dan VB6. Volgens Microsoft zijn de systeemeisen voor VS.NET minimaal 128 Mb. De praktijk wijst uit dat minimaal 256Mb nodig is. Pas dan presteert VS.NET weer zoals gewend bij VB6 met 128 Mb. (openen projecten, compileren etc.)

Tijdens de installatie van VS.NET wordt gecontroleerd welke Windows componenten benodigd zijn voor het gebruik van VS.NET. Hieraan is niets te kiezen, en de voorgestelde Windows componenten zijn dan ook allemaal nodig. De minimale componenten welke geïnstalleerd moeten worden zijn:
- Windows NT 4.0 Service Pack 6a;
- Windows 2000 Service Pack 2;
- Microsoft Windows Installer 2.0;
- Microsoft Windows Management Instrumentation;
- Microsoft FrontPage 2000 Web Extensions Client;
- Microsoft FrontPage 2000 Server Extensions Service Release 1.2;
- Setup Runtime Files;
- Microsoft Internet Explorer 6;
- Microsoft Data Access Components 2.7;
- Microsoft Jet 4.0 Service Pack 3;
- Microsoft .NET Framework.

6.2 Gebruiker PC

De eisen aan de PC van gebruikers zijn mede afhankelijk van de soort .NET applicatie. Bij Web applicaties zal de gebruiker moeten beschikken over een browser die .NET applicaties ondersteund. In principe worden alle browsers ondersteund door .NET. Echter deze ondersteuning geschiedt door het al dan niet genereren van bepaalde functionaliteit voor de browser. Het is dan ook aan te raden om Microsoft Internet Explorer 6 te gebruiken om

volledig gebruik te kunnen maken van .NET applicaties. Deze versie is standaard aanwezig in Windows XP, maar zal op Windows 2000 PC's geïnstalleerd moeten worden.
Bij een Desktop applicatie moet, zoals reeds eerder genoemd, de PC van elke gebruiker beschikken over de Common Language Runtime. De applicatie wordt, gecompileerd tot Intermediate Language, geïnstalleerd op de PC van de gebruiker. De Common Language Runtime compileert de applicatie tijdens het opstarten tot, door de computer te begrijpen, Native Code. De Common Language Runtime is onderdeel van het .NET Framework en kan gedownload worden vanaf:
http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/829/msdncompositedoc.xml
De PC van een gebruiker zal bij Desktop applicaties meer intern geheugen nodig hebben dan bij Web applicaties. Dit komt, omdat de Common Language Runtime de Desktop applicatie compileert tot native code. Bij Web applicaties gebeurt dit al op de web server. Aan te raden is om de PC's minimaal met 128 Mb uit te rusten.
Bij Web applicaties is eveneens de resolutie van het beeldscherm van belang. Zo zal een Web applicatie die gebouwd is op een resolutie 1024x768 niet compleet getoond worden indien de grafische kaart c.q. beeldscherm van de gebruiker maximaal 800x600 aan kan. Dit is uiteraard allereerst de verantwoording van de ontwikkelaar om een Web applicatie voor de minimaal aanwezige resolutie geschikt te maken. Om echter optimaal gebruik te maken van het beeldscherm wordt aangeraden alle PC's geschikt te maken voor een resolutie van 1024x768.

6.3 Servers

Voor de servers kunnen drie verschillende typen onderscheiden worden, te weten: de web server, de applicatie server en de database server.

6.3.1 Web server en applicatie server

Bij applicaties die binnen het MNP gebruikt worden kan volstaan worden om één PC zowel als web server als applicatie server te gebruiken. Het splitsen van web server en applicatie server wordt vooral gedaan om beveiligingsredenen. De implementatie van de communicatie tussen deze servers is echter complexer. Vandaar dat de voorkeur voor interne applicaties uit gaat naar één server. Deze PC moet minimaal Windows 2000 Professional als besturingssysteem hebben. Daarnaast moet de PC beschikken over Internet Information service 5 (IIS5).
Het geheugen is afhankelijk van de hoeveelheid Web applicaties die gedraaid worden op de web server, het aantal gebruikers en de hoeveelheid data die uit de onderliggende database wordt gehaald. Aangeraden wordt om de web server uit te rusten met 1Gb aan intern geheugen met een 1.4 GHz processor. Indien de web server en de Applicatie server gescheiden worden dan kan het beste twee gelijkwaardige machines gebruikt worden, zodat ze voor beide doeleinden geschikt zijn.

6.3.2 Database server

De systeemeisen voor een database server worden bepaald door de DBA van de server. Dit wordt in deze Reference Guide dan ook buiten beschouwing gelaten.

7. Conclusie

In het hoofdstuk 2 werd gesteld dat het MNP over moet gaan naar.NET. Daarna is vooral ingegaan op de inhoudelijke kant van .NET. Hierbij werd wel de kanttekening geplaatst dat het toepassen van .NET per applicatie bekeken moet worden. Om hier een goede keuze te maken is het van belang om zowel het nut als de toepasbaarheid van ,NET voor het MNP te bekijken. In dit hoofdstuk worden deze twee aspecten bekeken.

7.1 Algemeen

Nut

Het feit dat Microsoft met de .NET technologie een gehele nieuwe visie neer zet, zorgt voor een breed toepassingsgebied. De .NET technologie sluit aan bij behoeftes aan (web) applicaties en tegelijk aan het beschikbaar stellen van gegevens. Het MNP maakt zowel gebruik van Windows als van Office. Doordat de nieuwe versies van Windows als Office standaard de .NET technologie bevatten zal .NET in de toekomst goed aansluiten op de infrastructuur van het MNP. Daarnaast is Microsoft druk bezig om ook de aansluiting te maken naar andere platforms zoals bijvoorbeeld Unix en Oracle. Een groot voordeel voor het MNP is dan ook dat met behulp van .NET allerlei systemen en applicaties kunnen samenwerken en gebruik kunnen maken van dezelfde gegevens. Dit betekent onder andere dat .NET goed geïmplementeerd kan worden in de infrastructuur zoals deze is beschreven in de A&K analyse.

Een ander feit is dat het .NET platform vele ontwikkeltalen ondersteunt. Dit zorgt ervoor dat ontwikkelaars met diverse achtergronden toch samen een applicatie kunnen ontwikkelen. Zo zal een modelleur een rekenmodel in Fortran.NET kunnen maken, waarna een VB programmeur een User Interface bouwt met behulp van ASP.NET. Zo kan het rekenmodel, met behulp van een rijke interface, voorzien worden van de juiste parameters en kan de uitkomst van het rekenmodel grafisch worden weergegeven.

Met het voorgenoemde voorbeeld kan gelijk een ander nut van .NET worden genoemd. Doordat met behulp van .NET gemakkelijk een rijke web interface gebouwd kan worden, kan ook een RIVM applicatie of model gemakkelijk buiten het RIVM gebruikt worden. Immers voor een web applicatie heeft de gebruiker niets anders nodig dan een web browser. Inmiddels heeft dit zich in het .NET project eMJV van het MNP reeds bewezen.

Tot slot is met de komst van assemblies niet alleen de DLL-hell opgelost, maar is ook de distributie eenvoudiger geworden. Bij Desktop applicaties is echter wel het Framework nodig om de applicatie te kunnen gebruiken. Bij Web applicaties is de distributie nog eenvoudiger. Hier hoeft de applicatie en het Framework slechts op de web server geïnstalleerd te worden. Zowel de assemblies als de distributie bieden grote voordelen bij het beheren en uitrollen van applicaties.

Toepassing

In de vorige paragraaf is geschetst dat .NET toegevoegde waarde heeft voor het MNP. Het is echter ook van belang dat .de NET technologie toepasbaar is voor het MNP. Om te beginnen zijn er echter twee grote nadelen te noemen over .NET. Ten eerste is er op dit moment weinig kennis over .NET binnen het MNP. Ten tweede geldt dat .NET erg uitgebreid is en dat het daarom ook niet met een cursusje te leren valt. Het Framework biedt zoveel functionaliteit dat het lastig kan zijn om bepaalde functies te vinden.

Toch loont het zeker de moeite om te investeren in een opleiding om de .NET technologie eigen te maken. Het feit dat het Framework zoveel functionaliteit biedt is namelijk ook een groot voordeel. Hierdoor hoeft er weinig geprogrammeerd te worden. Dit betekent weer dat er sneller ontwikkeld kan worden.
Voor het ontwikkelen van applicaties kan de ontwikkelaar gebruik maken van de gebruiksvriendelijke IDE van Microsoft.NET. Hiermee kan op een eenvoudige manier een rijke User Interface gemaakt worden. Zoals reeds gewend van de ontwikkelomgeving van Microsoft kan voor Desktop applicaties vrij eenvoudig een User Interface gebouwd worden. In .NET is het aantal standaard controls voor deze User Interface nog eens sterk uitgebreid. Deze ontwikkelomgeving kan nu ook gebruikt worden voor het ontwikkelen van web applicaties. Ook voor web applicaties geldt dat het aantal controls sterk is uitgebreid. Desondanks zullen er controls gemist worden die wel aanwezig zijn voor Desktop applicaties.

Daarnaast is de nadruk gelegd op het scheiden van programmeer code en de opmaak van applicaties. Hiermee worden applicaties schaalbaar en onderhoudbaar. Naast het feit dat de distributie een stuk eenvoudiger is geworden, zorgt dit ervoor dat .NET applicaties gemakkelijk in te zetten zijn binnen het MNP. Het uitwisselen van code tussen ontwikkelaars zorgt voor een uniformere manier van programmeren. Dit wordt nog eens versterkt door het feit dat .NET veel programmeertalen ondersteunt. Het gebruik maken van de kennis van andere ontwikkelaars is dus niet zo zeer meer verbonden aan een programmeertaal. Een Fortran.NET programmeer zal namelijk het Framework op dezelfde manier gebruiken als een VB.NET programmeer. Alleen zullen ze het in een andere syntax schrijven.

7.2 Nut van .NET technologie

Hoofdstuk 0 is begonnen met de kreet ‘Connected Software’ van Microsoft, om vervolgens aan te geven dat er veel mogelijk is met .NET. Verschillende devices kunnen data uitwisselen met Servers over de hele wereld. De praktijk is echter toch wat anders. Ten eerste zijn voor veel bedrijven en ook voor het MNP de mogelijkheden om bijvoorbeeld PDA’s aan te sturen niet interessant. Ook al zullen er altijd fanatieke onderzoekers zijn die een update van de emmissie gegevens ge-sms’d willen hebben om die vervolgens in grafiek vorm op hun PDA te zien. Erg waarschijnlijk is het niet dat dit zal worden ontwikkeld.

7.2.1 .NET is Multilingual

Toch heeft .NET zeker nut voor het MNP. Allereerst sluit .NET aan bij veel ontwikkeltalen. Binnen één .NET solution is het mogelijk om gebruik te maken van meerdere talen. Zo is het mogelijk dat een C, Delphi, Java en VB ontwikkelaar samen werken aan één applicatie. Alle talen maken gebruik van één IDE, één Debugger en niet onbelangrijk, één Framework. Het .NET Framework biedt veel standaard functionaliteit die de ontwikkeltijd van applicaties verkleint.

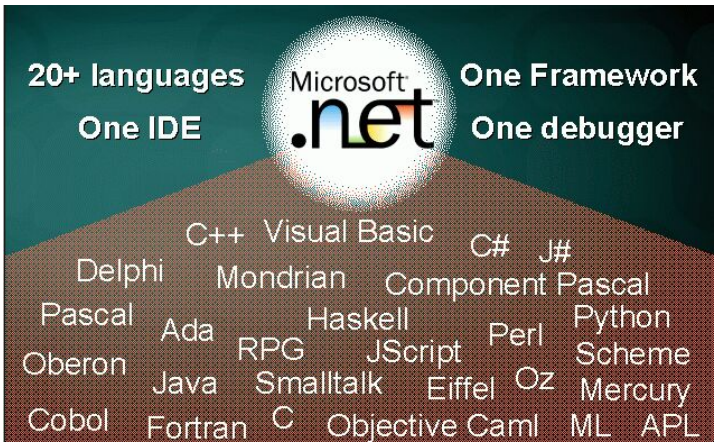


Fig. 11: .NET is multilingual

7.2.2 Distributie

Door de komst van assemblies is de distributie van .NET applicaties een stuk eenvoudiger geworden. In principe kunnen de assemblies van een applicatie gekopieerd worden vanaf de ontwikkel PC naar de gebruiker PC, c.q. de webserver. DLL's hoeven niet meer geregistreerd te worden, omdat alle informatie die voorheen in het register stond, nu aanwezig is in de metadata van de assembly. Verder is het mogelijk om een nieuwere versie van een DLL naast een reeds bestaande versie te installeren. Dit voorkomt de veel gevreesde DLL-Hell.

Voor Desktop applicaties moet eenmalig het .NET Framework geïnstalleerd worden op de gebruiker PC.
In principe kunnen de assemblies van een Desktop applicatie met behulp van de copy-paste methode op de gebruiker PC geplaatst worden. Alleen DLL's die in de Global Assembly Cache geplaatst worden vragen een uitgebreidere setup. Dit wordt daarom afgeraden, het is beter om de DLL's in de directory of sub directory van de applicatie te houden.

Een groot voordeel van Web applicaties is dat hier in het geheel geen distributie meer aan te pas komt, anders dan dat de bestanden op de webserver worden geplaatst. De gebruiker kan middels een email of andere communicatie op de hoogte worden gebracht met welke URL de applicatie kan worden opgestart. Ook kan met behulp van Novell een link geplaatst worden in het start menu.

7.2.3 Web Services

Web Services is een geheel nieuwe technologie die met .NET geïntroduceerd wordt. Het is de algemene verwachting van grote ICT bedrijven dat Web Services de nieuwe manier wordt om remote via het web met andere DLL's te communiceren. Een andere verwachting is dat Web Services de manier wordt om kennis te delen. In dit geval wordt onder kennis ook wel functionaliteit verstaan. De Web Services zijn in dit geval een gecompileerde componenten bibliotheek.

In eerste instantie zal het nut van Web Services voor het MNP zich nog moeten bewijzen. Zo kan het handig zijn om functionaliteit, zoals het interpreteren en verwerken van emissie data, door middel van een Web Service aan te bieden. Deze zou dan MNP breed in alle projecten die gebruik maken van deze database server gebruikt kunnen worden. Dit is onafhankelijk van de ontwikkeltaal van het project. Het gevaar bestaat echter wel dat zo'n Web Service een

bepaalde functionaliteit biedt die niet naadloos aansluit op de wens van de ontwikkelaar. Als conclusie kan in elk geval getrokken worden dat Web Service niet te veel functionaliteit moeten bevatten. Hierdoor wordt een te grote afhankelijkheid gecreëerd van code van een andere ontwikkelaar.

Web Services binnen een project kunnen wel voordeel hebben, indien de applicatie over meerdere servers verdeeld is. In dit geval wordt de Web Services hoofdzakelijk gebruikt om code remote te kunnen benaderen.

7.3 Toepasbaarheid .NET binnen MNP

De toepasbaarheid van de .NET technologie moet voor drie typen applicaties bekeken worden, te weten stand-alone Desktop applicaties, (eenvoudige) Web applicaties en Enterprise applicaties.

Voordat er ingegaan kan worden op de diverse typen applicaties is het van belang om te kunnen bepalen wanneer welk type applicatie nuttig is.
Factoren die hierbij een rol spelen zijn:
- nieuwbouw of bestaande applicatie;
- aantal wijzigingen op bestaande applicatie;
- het aantal gebruikers;
- de levensduur van de applicatie.

Bij nieuwbouw applicaties speelt eigenlijk alleen de onbekendheid van ontwikkelaars met .NET een rol bij de afweging. Een argument om bestaande ontwikkelmethodes te gebruiken kan zijn dat er een kleine applicatie gebouwd moet worden voor een enkele gebruiker in een zeer korte doorlooptijd.
In het begin zal het ontwikkelen met .NET meer ontwikkeltijd vragen. Dit zal op den duur verdwijnen en zelfs ten gunste van .NET vallen indien de gebruiker meer kennis opdoet van de programmeertalen in .NET en van het Framework in het bijzonder. Vooral dit laatste kan leiden tot reductie van de ontwikkeltijd, aangezien het framework veel functionaliteit bevat die voorheen zelf gebouwd moest worden.
Over het algemeen kan de conclusie getrokken worden dat voor nieuwbouw .NET zeer geschikt is.

Bij bestaande applicaties spelen een aantal andere factoren een rol. Het converteren van een bestaande applicatie kost veel tijd. Daarnaast zal door de conversie wizard veel code geconverteerd worden naar de VB6 compatibility Class. Hierdoor maakt de applicatie niet gebruik van de voordelen van de nieuwe .NET technologie. Er kan dus geconcludeerd worden dat het niet aan te raden is om bestaande applicaties teconverteren.
Toch zijn er een aantal redenen om toch te besluiten een bestaande applicatie te converteren. Dit zijn:
- Er moet veel functionaliteit gewijzigd worden (deze wijzigingen staan bijna gelijk aan nieuwbouw);
- Een Desktop applicatie moet opgewaardeerd worden naar een Web applicatie;
- Het aantal gebruikers van een applicatie is enorm toegenomen;
- De huidige applicatie is niet onderhoudbaar;
- Er zijn grote problemen met de distributie van de huidige applicatie.

In de volgende tabel staan de voor- en nadelen weergegeven tussen de ‘oude’ programmeertaal en het .NET equivalent voor zowel Desktop als Web applicaties.

Tabel. 6: Voor- en nadelen bij Desktop en Web applicaties

	VB6	VB. NET	ASP. NET
Voordelen	hergebruik bestaande applicaties	gebruik Framework	gebruik Framework
	rijke UserInterface	nog rijkere UserInterface	Web enabled
	gebruik lokale databases	gebruik lokale databases	tunen performance
	huidige kennis	gebruik Web Services	gebruik Web Services
		eenvoudige distributie	copy-paste distributie
		ondersteuning XML	ondersteuning XML
Nadelen	distributie	kennisopbouw nodig	veel kennis nodig
	matige ondersteuning XML	in combinatie met Web Services of oracle niet lokaal te gebruiken	niet off-line te gebruiken
	DLL-Hell		langere ontwikkeltijd
			geen gebruik van lokale databases
			minder rijke User Interface

7.3.1 Stand-alone Desktop applicaties

Voor kleine applicaties die voor een enkele gebruiker beschikbaar moeten zijn, is een VB.NET applicatie het meest geschikt. Op het moment dat het .NET Framework via de NAL van Novell beschikbaar is, dan is de distributie van een VB.NET Desktop applicatie geen probleem meer. De ontwikkelaar hoeft dan alleen de gecompileerde bestanden van de applicatie via de NAL te laten distribueren.

Ook is het mogelijk om met een .NET Desktop applicatie gebruik te maken van de functionaliteit die de gebruikers PC biedt. Dit zijn onder andere het gebruik van een lokale database, Excel functionaliteit, Word functionaliteit en het gebruik van het register van de gebruikers PC. Dit laatste kan gebruikt worden om gebruiker specifieke gegevens in op te slaan.

Indien een applicatie door veel gebruikers gebruikt wordt, kan toch de voorkeur uitgaan naar een Web applicatie.

7.3.2 Web applicaties

Binnen het MNP wordt het standaard om alle nieuwe applicaties Web enabled te maken. Als eerste moet geconcludeerd worden dat VS.NET uitstekend geschikt is voor het maken van web applicaties. De Class Library ASP.NET is een grote stap vooruit ten opzichte van de oude ASP technologie. Dit komt onder andere doordat ASP.NET de RAD methode ondersteunt. Hierdoor kan in een vrij korte tijd een eenvoudige web User Interface gemaakt worden.
Er moet echter wel rekening mee gehouden worden dat web applicaties nog steeds meer ontwikkeltijd (ongeveer een factor 2) vragen dan desktop applicaties. Deze factor kan nog verder oplopen waneer er schrijf applicaties ontwikkeld worden, en er controls worden gebruikt die niet standaard aanwezig zijn voor het web.

Verder moet er rekening gehouden worden met het gebruik van state en met performance issues. Afgezien van een andere aanpak c.q. ontwerp voor Desktop applicaties, hoeft dit geen bezwaar te zijn.

De applicatie zal getest moeten worden met veel simultane gebruikers. De situatie tijdens het ontwikkelen geeft niet de situatie weer zoals deze in productie zal zijn. De webserver moet dan namelijk de requests van veel gebruikers afhandelen. Dit in tegenstelling tot Desktop applicaties, waar de lokale applicatie door één gebruiker wordt gebruikt.

Ondanks dat het aantal controls is toegenomen ten opzichte van ASP, blijft de rijkheid van de User Interface sterk achter bij Desktop applicaties. Het aanschaffen van zogenaamde 3rd party controls zorgt voor een grote afhankelijkheid van andere bedrijven. Er moet dan ook goed worden bekeken of de functionaliteit die de control biedt niet op een andere manier gerealiseerd kan worden. Eén van de grootste risico's van 3rd party controls is dat ze niet meer goed functioneren bij een update van het .NET Framework. Het kan een alternatief zijn als het zelf ontwikkelen teveel ontwikkeltijd vergt. Enige gereserveerdheid ten opzichte van 3rd party controls is dus op zijn plaats.

7.3.3 Enterprise applicaties

Enterprise applicaties worden gekenmerkt door een complexere infrastructuur dan desktop applicaties en eenvoudige web applicaties. Er moet rekening gehouden worden met het inrichten van één of meerdere servers en ook beveiliging (zeker bij internet applicaties) gaat een rol spelen.

De .NET technologie is zeer geschikt voor het bouwen van Enterprise applicaties. Het biedt goede ondersteuning om diverse servers aan te spreken. Ook beveiliging is standaard aanwezig binnen het Framework. Het is daarom aan te raden om goed na te denken over de beveiliging alvorens te gaan programmeren.

7.3.4 Databases

Het Framework bevat standaard Data Providers voor SQL server, OleDb en ODBC. Verder is er nu ook een Native Provider voor Oracle op de markt. Dit zorgt voor een goede integratie van .NET applicaties met SQLServer, Oracle, Access of andere databases die OleDb en ODBC ondersteunen.

7.4 Vereiste kennis

Op het moment dat een ontwikkelaar een .NET applicatie gaat bouwen, is het nodig dat hij bepaalde kennis heeft van de .NET technologie.

Alhoewel de syntax binnen de ontwikkeltalen van VS.NET voor iedere ontwikkelaar goed te lezen zal zijn, zijn er toch een groot aantal veranderingen doorgevoerd. Deze veranderingen vragen een andere manier van programmeren, en kan soms tot frustraties leiden, waarbij teruggrijpen naar de 'oude' vertrouwde ontwikkelmethodes soms aantrekkelijk lijkt. Het is nog steeds mogelijk om recht-toe-recht-aan te blijven programmeren, maar het verdient de aanbeveling om bij nieuwe applicaties toch zoveel mogelijk van deze nieuwe technologie gebruik te maken. Deze nieuwe technologie heeft zoveel meer te bieden dat het zonde is ze niet te gebruiken. Goede kennis van het Microsoft .NET Framework is hierbij onontbeerlijk.

In de volgende tabel wordt uitgegaan van een VBA / VB6 ontwikkelaar en de kennis die deze ontwikkelaar nodig heeft om over te kunnen stappen naar .NET. Er is onderscheid gemaakt tussen vereist, gewenst en handig. Met ‘vereist’ wordt bedoeld dat zonder deze kennis de ontwikkelaar niet in staat is om een .NET applicatie te bouwen. Onder ‘gewenst’ wordt verstaan dat het zeer nodig is dat de ontwikkelaar op de hoogte is van het bestaan en eventueel voorbeelden uit help bestanden of van het Internet kan begrijpen en implementeren. Onder ‘handig’, wordt verstaan dat de ontwikkelaar geen kennis van deze technologie hoeft te hebben, maar dat het wel voordelen biedt indien de ontwikkelaar hiervan kennis heeft.

Tabel. 7: Benodigde kennis

Kennis	Desktop applicaties	Web applicaties	Enterprise applicaties
Object Oriented Programming (OOP)	vereist	vereist	vereist
VB.NET	vereist	vereist	vereist
ASP. NET		vereist	vereist (bij web UI)
ADO.NET	vereist	vereist	vereist
Exception handling	vereist	vereist	vereist
.NET Framework	gewenst	gewenst	gewenst
XML	gewenst	gewenst	vereist
HTML		vereist	vereist (bij web UI)
Cascading Style Sheets		gewenst	gewenst (bij web UI)
Javascript		gewenst	gewenst (bij web UI)
Assemblies	handig	handig	handig
Architectuur (multi tier)	handig	gewenst	vereist
Web Services	bij remote applicaties gewenst		vereist
Internet Information Service		handig	gewenst
HTTP protocol		handig	handig

7.5 Aanbevelingen voor verder onderzoek

Ondanks dat geprobeerd is om veel onderdelen van Microsoft.NET te behandelen, is het onmogelijk om alles te bespreken en te verduidelijken. Er zijn dan ook een aantal onderwerpen die zeker de moeite waard zijn om verder te onderzoeken. In deze paragraaf zullen een aantal aangehaald worden.

.NET Remoting

Met Web services is het mogelijk om applicatie logica remote te benaderen via het HTTP protocol. Het Framework biedt ook mogelijkheid om gebruik te maken van .NET Remoting. Dit is de opvolger van COM. De voordelen van .NET remoting ten opzichte van Web Services is dat het veel sneller is, met name doordat het gebruik maakt van het TCP/IP protocol in plaats van het HTTP protocol. Nadeel is dat het niet via het Web toegankelijk is en dat daarom de computers altijd rechtstreeks verbonden moeten zijn. Binnen een lokaal netwerk is het echter de moeite om .NET Remoting nader te bestuderen. Hierbij kan gedacht worden aan hoe een .NET Remote server ingericht moet worden, hoe een aanroep vanaf een client gerealiseerd kan worden en of er migratie mogelijkheden zijn om van een Web Service naar .NET Remoting en visa versa te gaan.

Office XP

Excel in Office XP biedt een aantal web controls die geïntegreerd kunnen worden in een browser. Excel biedt met pivot tables en grafieken een vorm van OLAP. Het kan daarom aantrekkelijk zijn om Office componenten in een web applicatie te gebruiken. Volgens Microsoft zal de opvolger van Office XP (te weten Office 2003) alle .NET ontwikkeltalen als primaire ontwikkeltaal ondersteunen. Hiermee komt er een einde aan het gebruik van VBA als programmeertaal.

Visio

Visio is een pakket waarmee o.a. ontwerpen van applicaties gemaakt kunnen worden. Hierbij kan gedacht worden aan UML schetsen, maar ook aan het ontwerp van specifieke classes. Visio 2002 integreert met VS.NET. Door deze integratie kan het onder andere van Classes UML schema's genereren.

XML Serialization

XML Serialization is een methode om gemakkelijk XML bestanden te genereren en te lezen. Het is hiermee mogelijk om allerlei bestandsformaten om te vertalen naar XML. Het is echter ook mogelijk om complete objecten en hun inhoud om te zetten naar XML en deze ergens anders weer terug te zetten naar objecten. Dit kan bijvoorbeeld nuttig zijn bij het versturen van data over het netwerk.

Bovenstaande onderwerpen zijn zeker de moeite waard om verder te bestuderen en onderzoeken hoe deze toegepast kunnen worden.

Termen / afkortingen

Term / afkorting	Korte beschrijving	Betekenis	Pagina
2-tier		Applicatie bestaande uit 2 lagen, te weten de client-applicatie met geïntegreerde data laag en een database	15, 16
3rd party		Software / code van externe organisaties die gebruikt wordt voor eigen applicaties	69, 82
3-tier		Applicatie bestaande uit 3 lagen, te weten de client-applicatie, een aparte data laag en een database	15, 16
ActiveX		Technologie om o.a. interactieve informatie aan te bieden op Inter- / intranet	14, 30, 34, 35, 36, 38, 46, 54
ADO	ActiveX Data Objects	Library met functies om databases mee te benaderen	14, 16, 44, 46, 47, 48, 49, 53, 54
ADO.NET		Class library in het .NET Framework, opvolger van ADO	14, 19, 22, 46, 47, 48, 49, 65, 66, 67, 83
API	Application Program Interface	Interface met functies van een programma of besturingssysteem, die door andere programma's gebruikt kan worden	22, 48, 70, 71
ASCX		Extentie van een ASP.NET user control	38
ASMX		Extentie van een ASP.NET Web Service	61
ASP	Active Server Pages	Techniek om dynamische web applicaties te ontwikkelen met behulp van Javascript of VBscript	4, 34, 36, 37, 38, 39, 63, 71, 82
ASP.NET		Nieuwe versie van ASP voor het .NET platform	4, 8, 13, 14, 19, 21, 23, 30, 36, 37, 38, 39, 40, 63, 69, 70, 71, 72, 75, 77, 81, 83
ASPX		Extentie van een ASP.NET Web Form	60, 63
Assembly		Tot MSIL gecompileerd .NET project (DLL of .EXE)	3, 7, 19, 20, 30, 79
Attribuut		Een attribuut wordt gebruikt om een XML element specifieker te beschrijven, voorbeeld ('belang' is de attribuut): <element belang="hoog">Tekst</element>.	41
Bound		In combinatie met control: een control die rechtstreeks gekoppeld is aan een Recordset of Dataset. De data uit de Recordset of Dataset wordt automatisch gevuld in de control	39
Browser		Software om web applicaties via het HTTP protocol te gebruiken	4, 20, 44, 46, 60, 61, 71, 72, 73, 75, 77
Business laag		Laag in een n-tier applicatie die de implementatie van de business-rules bevat	14, 60, 63, 65
C#	Uitspraak: Sie Sjarp	Nieuwe ontwikkeltaal voor .NET die is afgeleid van C	4, 7, 8, 13, 21, 24, 30, 37, 52, 55, 69

Cache		Cache is het gedeelte van het geheugen waar de laatst benaderde data in staat	50
Class		Een class is een abstracte beschrijving c.q. implementatie van een aantal variabelen en methoden	26, 27, 28, 29, 31, 35, 37, 54, 55, 63, 64, 65, 66, 80, 84
Class Library		Verzameling gerelateerde classes	3, 19, 21, 22, 23, 30, 37, 39, 69, 70, 71, 81
Client		Gebruikers PC, ook wel de applicatie die op de gebruikers PC draait	19, 20, 22, 36, 44, 60, 63, 65, 69, 71, 72, 84
CLR	Common Language Runtime	Compiler en tools voor het uitvoeren van .NET applicaties	19, 20, 21, 24
COM	Component Object Model	Programmeermodel om objecten en componenten met elkaar te laten communiceren, bestaande uit COM+, DCOM en Active X	31, 48, 49, 83
Constructor		Instantiatie van object	26, 27
Cookie		Informatie die door een web applicatie kan worden weggeschreven en uitgelezen op de client	72, 73
CSS	Cascading Style Sheet	Bestand dat een beschrijving bevat van de opmaak van een HTML document	40
Cursor		Onderdeel van ADO dat de methode bevat hoe een applicatie dient om te gaan met de rijen in een Recordset	47
DAO	Data Access Objects	Library om een Microsoft Access database te benaderen	54
Data laag		Laag in een n-tier applicatie die de implementatie van de communicatie met een database bevat	14, 60, 63, 65, 67
Data Provider		.NET Library om databases te benaderen	51, 52, 82
DataAdapter		Class uit een DataProvider om een tabel in een Dataset aan te sturen	47, 48, 49, 50, 65
DataReader		In het geheugen opgebouwde alleen-lezen tabel	47, 48, 49
DataSet		In het geheugen opgebouwde database bestaande uit één of meerdere tabellen welke gekoppeld kunnen worden via relaties	39, 47, 48, 49, 50, 71
DataTable		Tabel in een dataset	47, 50
DCOM	Distributed COM	Services die gebruikt kunnen worden om COM objecten remote met elkaar te laten communiceren	61, 63
Delphi.NET		Ontwikkeltaal van Borland die samen werkt met het .NET Framework	21, 23
Design mode		Status van een applicatie zoals de ontwikkelaar deze ziet tijdens het ontwerpen	51
Desktop applicatie		Windows applicatie die op de PC van de gebruiker draait	7, 8, 9, 20, 23, 37, 39, 60, 65, 69, 70, 71, 76, 77, 78, 79, 80, 81, 82, 83

DHTML	Dynamic HTML	Verzamelnaam voor dynamische Web pagina's gemaakt met behulp van Javascript en / of ActiveX controls	35, 36
DISCO		Concept uit UDDI die beschrijft hoe Web Services met behulp van discovery documenten en het discovery protocol opgezocht kunnen worden	61
Discovery document		Een XML bestand dat koppelingen naar andere discovery bestanden, XSD schema's en Web services bevat. Een discovery bestand is nodig bij het opzoeken en uitvragen van een Web service	61
Discovery protocol		Protocol dat gebruikt wordt voor het opzoeken en uitvragen van informatie over Web services	61
DLL	Dynamic Link Library	Gecompileerde code die door één of meerdere programma's kan worden aangesproken en alleen in het geheugen aanwezig is tijdens deze aanroep	3, 7, 19, 21, 29, 30, 31, 34, 38, 59, 61, 71, 79
DLL-hell		Alle nadelen van DLL's (zoals problemen met verschillende versies en het moeten registreren van DLL's) worden vaak de DLL-hell genoemd	3, 5, 9, 20, 77, 79, 81
DOM	Document Object Model	Model om XML documenten gestructureerd vanuit code te benaderen	44
DTD	Document Type Definition	Manier om de definitie van XML elementen en attributen te beschrijven	41, 42, 43
Element		Een element komt uit de MarkUp language en staat om de data heen, voorbeeld ('zin' is het element): <zin belang="hoog">Tekst</zin>.	41, 42, 43, 44, 47, 54
Exception Handling		Foutafhandeling binnen .NET	4, 32, 83
EXE	Executable	Gecompileerde code die zelfstandig kan opereren in het geheugen van een IBM compatible PC	19, 21, 29, 30, 34, 35
FW	Framework	CLR en een set Class Libraries die een groot hoeveelheid functies bevatten	3, 4, 5, 7, 8, 9, 16, 18, 19, 20, 22, 23, 31, 34, 35, 36, 37, 42, 46, 51, 52, 60, 65, 70, 75, 76, 77, 78, 79, 80, 81, 82, 83
GAC	Global assembly cache	Centrale locatie op een PC waarin alle assemblies staan die door meerdere applicaties gebruikt mogen worden	20, 21, 51, 79
GC	Garbage Collector	Automatisch beheren en opruimen van objecten in het geheugen	31
GUI	Graphical User Interface	De grafische weergave van een applicatie waarmee de gebruiker de applicatie aanstuurt	60, 63, 65
Header		Niet zichtbaar gedeelte van een internet pagina waarin gegevens staan vermeld die	38, 62

		door de webserver worden gebruikt	
HTML	Hyper Text Markup Langage	Taal om web pagina's te maken	4, 8, 20, 36, 37, 38, 42, 44, 48, 63, 69, 70, 71, 72, 83
HTTP	HyperText Transfer Protocol	Protocol om gegevens over het web te sturen	44, 60, 61, 62, 63, 72, 73, 83
IDE	Integrated Development Environment	Ontwikkelomgeving om applicaties in te ontwikkelen	13, 23, 32, 34, 40, 52, 55, 66, 71, 78
IIS	Internet Information Server	Webserver van Microsoft die gebruikt kan worden op een PC met een Windows besturingssysteem	63, 75
Inheritance		OO concept waarvan een subclass de properties en methods van een Class kan erven	26, 28, 29
Interface		Class waarbij de properties en methods gedefinieerd zijn maar die geen code heeft	17, 19, 20, 26, 29, 31, 38, 44, 66, 77, 78
JIT	Just In Time	Methode waarbij MSIL code wordt gecompileerd tot Native Code op het moment dat de gebruiker de applicatie opstart	19
LDAP	Light Directory Access Protocol	Protocol om objecten in een netwerk te lokaliseren	60
MDAC	Microsoft Data Access Components	Verzameling van libraries om databases te benaderen	44
Metadata		Gedeelte van een assembly dat informatie over de gebruikte base classes, interfaces en de methods, fields, properties en events van de assembly beschrijft	19, 20, 40, 42, 79
Module		Verzameling van functies in een applicatie. De module hoeft in tegenstelling tot een class niet geïnitialiseerd te worden om de functies te kunnen gebruiken	24, 26, 37, 70
MSDN library		Online naslagwerk van Microsoft op het gebied van software ontwikkeling	22
MSIL	Microsoft Intermediate Language	Taal die door de CLR van .NET geïnterpreteerd kan worden. Alle .NET ontwikkeltalen (ook niet Microsoft talen) compileren naar MSIL	19
Namespace		Naam die een groep namen uniek maakt, bij XML gaat het om de namen van elementen en attributen, bij Classes gaat het om de naam van de classes	30, 31, 42, 51, 52, 70
Native Code		Taal waarnaar alle Windows applicaties worden gecompileerd, zodat ze door de computer geïnterpreteerd en uitgevoerd kunnen worden	19, 60, 76
Native Provider		Data Provider die geschikt is gemaakt voor één soort database	51, 65, 66, 82

n-tier		Applicatie die bestaat uit meerdere lagen, hetzij fysiek hetzij logisch gescheiden	3, 8, 15, 16, 22
OLAP	On-line Analyzing Processing	Methodiek om snel en gemakkelijk inzicht te krijgen in meer dimensionale data	84
OO	Object Oriented	Object geOriënteerd (zie ook OOP)	25
OOP	Object Oriented Programming	Concept om op basis van objecten applicaties te ontwikkelen	34, 83
Overloading		Concept waarbij meerdere functies met dezelfde naam, maar met andere of meer parameters gebruikt kunnen worden	26, 27, 53
Parameters		Extra waardes die meegegeven worden aan een functie	52
PDA	Personal Digital Assistent	Handformaat computer, geschikt als digitale agenda, meest bekend zijn Palm en Pocket PC	17, 78
PE	Portable Execution	Executable gecompileerd naar MSIL	19
PHP	oorspronkelijk Personal Home Page nu PHP: Hypertext Preprocessor	Ontwikkeltaal voor het ontwikkelen van web applicaties, o.a. geschikt voor Unix	36
Property		Eigenschap van een class	29, 33, 40, 53, 54
Registry		Gedeelte van een Windows besturingssysteem (vanaf Win95) dat gebruikt wordt voor opslag van o.a. configuratie informatie over dll's, componenten en instellingen van applicaties	20, 69
Server		Programma die services aanbiedt aan andere programma's op één of meerdere PC's. De PC waarop dit programma draait wordt vaak ook wel een server genoemd	5, 9, 15, 18, 19, 36, 37, 38, 46, 48, 59, 60, 63, 65, 66, 67, 69, 71, 72, 73, 75, 76, 79
Service		Een dienst die door een applicatie wordt aangeboden	60, 61, 62, 63, 64, 75, 76, 83
Service Pack		Verzameling updates voor software, bevat oplossingen voor bugs en eventueel nieuwe toevoegingen aan het programma	21, 75
Session		ASP(.NET) object waarmee gegevens in het geheugen van de webserver per gebruiker kunnen worden opgeslagen	71, 72, 73
Signature		De manier waarop een functie aangesproken kan worden bij overloading	27
SOAP	Simple Object Access Protocol	Protocol dat geschikt is om Web Services met elkaar te laten communiceren	18, 42, 62, 63
State		De informatie van een programma op een willekeurig tijdstip	71, 72, 82
Statefull		Type programma die informatie bij houdt van interacties die worden gedaan,	39, 65

		meestal door het bewaren van waarden in het geheugen	
Stateless		Type programma die geen informatie bij houdt van eerdere interacties, elke interactie dient volledig afgehandeld te kunnen worden aan de hand van de gegevens die tijdens de interactie worden verstrekt	72
Trace		Opsporen of volgen van data	38, 39
Type		Indeling voor variabelen waarbij het type aangeeft hoeveelheid geheugen gereserveerd dient te worden en wat voor soort informatie die variabele bevat	23, 24, 25, 26, 41, 43, 52
UDDI	Universal Description, Discovery and Integration	Concept om Web Services te vinden (‘gouden gids’)	18, 61, 62
UDT	User Defined Types	Combinatie van meerdere types in één variabele (beschikbaar in VB6)	29
UI	User Interface	De buitenkant van een applicatie zoals deze door een gebruiker wordt waargenomen	3, 4, 5, 8, 9, 14, 21, 23, 59, 60, 63, 64, 65, 69, 71, 77, 78, 81, 82, 83
UML	Unified Modelling Language	Generieke tekentaal om van een ‘real world’ concept een vertaling te maken naar een applicaties	19, 63, 64, 84
Unbound		In combinatie met control: een control die niet rechtstreeks gekoppeld is aan een Recordset of Dataset, maar waarbij de data via code toegekend wordt	39
User Interface laag		Laag in een n-tier applicatie dat de implementatie van de code voor de GUI bevat	60
User Interface		zie UI	3, 4, 5, 8, 9, 14, 21, 23, 59, 60, 63, 64, 65, 69, 71, 77, 78, 81, 82, 83
VB	Visual Basic	Ontwikkeltaal voor desktop applicaties	3, 14, 39, 44, 55, 77, 78
VB.NET	Visual Basic .NET	VB ontwikkeltaal in .NET	4, 7, 8, 13, 14, 21, 23, 24, 25, 26, 29, 30, 31, 32, 34, 35, 36, 37, 39, 46, 52, 55, 60, 69, 70, 71, 78, 81, 83
Viewstate		Oplossing van asp.net om de inhoud van de controls op te slaan.	72
VS	Visual Studio	Verzameling van Microsoft ontwikkeltalen en hun IDE	8, 13, 18, 19, 21, 23, 30, 32, 34, 36, 40, 52, 65, 75, 81, 82
VSS	Visual Source Safe	In Visual Studio geïntegreerde applicatie voor versiebeheer van applicatie code	36
W3C	World Wide	Het W3C is een industrie consortium dat	18, 40, 43, 44, 61,

	Web Consortium	als doel heeft om zoveel mogelijk standaarden voor het Web en de relaties tussen Web software te promoten. Ondanks dat het W3C is opgericht door grote multinationals is het toch 'merkloos' en zijn al hun producten gratis voor iedereen te verkrijgen	62
Web applicatie		Applicatie die draait op een web server, maar door gebruikers via een browser te benaderen is	7, 8, 9, 13, 14, 18, 19, 20, 21, 23, 36, 39, 46, 59, 60, 63, 69, 71, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84
Web server		Programma dat het HTTP en FTP protocol ondersteunt om Web applicaties mee te hosten	3, 4, 7, 8, 9, 20, 38, 60, 61, 63, 65, 69, 71, 75, 76, 77
WS	Web Service	Programmeerbare applicatie logica die via standaard internet protocols als dienst toegankelijk is	3, 4, 7, 8, 13, 14, 17, 18, 19, 21, 30, 36, 59, 60, 61, 62, 63, 65, 75, 79, 80, 81, 83, 84
WSDL	Web Service Description Language	XML-beschrijving van een interface van een Web Service	62
XHTML		Webpagina gegenereert met een XML en een XSL bestand	42, 44, 45, 46
XML	Extended Markup Language	Gestructureerde taal om data te beschrijven	14, 16, 17, 18, 22, 36, 37, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 54, 60, 61, 62, 81, 83, 84
XSD	XML Schema Definition	Manier om de definitie van XML elementen en attributen te beschrijven	41, 43
XSL	Extensible Style Sheet	Bestand dat een beschrijving bevat van de opmaak van een XML document	44, 45, 46
XSLT	XSL Translation	Taal die gebruikt wordt om een XML en een XSL bestand te interpreteren	44

Literatuur

Literatuur gemarkeerd met (!) wordt aanbevolen voor het verkrijgen van meer achtergrond informatie.

Jonathan Pinnock et al (mei 2001) Professional XML 2nd Edition ISBN 1861005059

Rockford Lhotka et al (april 2002) Professional VB.NET, 2nd edition ISBN 1861007167

David Sussman et al (mei 2002) Beginning ASP.NET 1.0 with VB.NET ISBN 1861007337

(!) Microsoft (april 2002) Defining the Basic Elements of .NET
http://www.microsoft.com/net/basics/whatis.asp

Steven Levy (januari 2002) So What's This .NET Thing?
http://www.microsoft.com/technet/treeview/?url=/technet/columns/ednote/en0102p1.asp

(!) Language Changes in Visual Basic
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vaconDifferencesBetweenVB6AndVB7.asp

(!) Visual Basic .NET Technology Map
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/techmap_vbnet.asp

INFO: New Data Types in Visual Studio .NET
http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q311327&

(!) XML Tutorial
http://www.xmlfiles.com

(!) W3Schools Online Web Tutorials
http://www.w3schools.com

Microsoft Design Guidelines
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnamingguidelines.asp

(!) WhatIs.com (IT encyclopedie)
http://whatis.com

Verzendlijst

Belanghebbenden binnen het RIVM:
1. Jan Aben (LED)
2. Gert van Alphen (IMP)
3. Aldrik Bakema (RIM)
4. Michel Bakkenes (NLB)
5. Arthur Beusen (IMP)
6. Fokke de Boer (IMP)
7. Laurens Brandes (IMP)
8. Kit Buurman (IMP)
9. Ron van Dijk (IMP)
10. Anton van der Giessen (IMP)
11. Alexander Gijsen (KMD)
12. Maarten 't Hart (LDL)
13. Stefan Hoogendoorn (KIT)
14. Onno Knol (NLB)
15. Frits Kragt (LDL)
16. Bert Leekstra (IMP)
17. Wim van der Maas (IMP)
18. Wim Mol (LED)
19. Romuald te Molder (IMP)
20. Durk Nijdam (NMD)
21. Ton de Nijs (RIM)
22. Rutger Nugteren (VTV)
23. Mark van Oorschot (NMD)
24. Rineke Oostenrijk (KMD)
25. Georgios Panagiotakopoulos (KIT)
26. Kees Peek (NMD)
27. Lennert Pronk (IMP)
28. Rob Puijk (KIT)
29. Liesbeth Rentinck (SCA)
30. Bart Rietveld (IMP)
31. Frank van Rijn (LOK)
32. Coen Schilderman (RIM)
33. Jaap Slootweg (LED)
34. Peter de Smet (LED)
35. Gert-Jan Stolwijk (IMP)
36. Sandy van Tol (LDL)
37. Coen van Tooren (IMP)
38. Pascal de Vink (IMP)
39. Jannie Vos (IMA)
40. Rick Wortelboer (NLB)
41. Mels de Zeeuw (KIT)
42. Bureau Rapportenregistratie